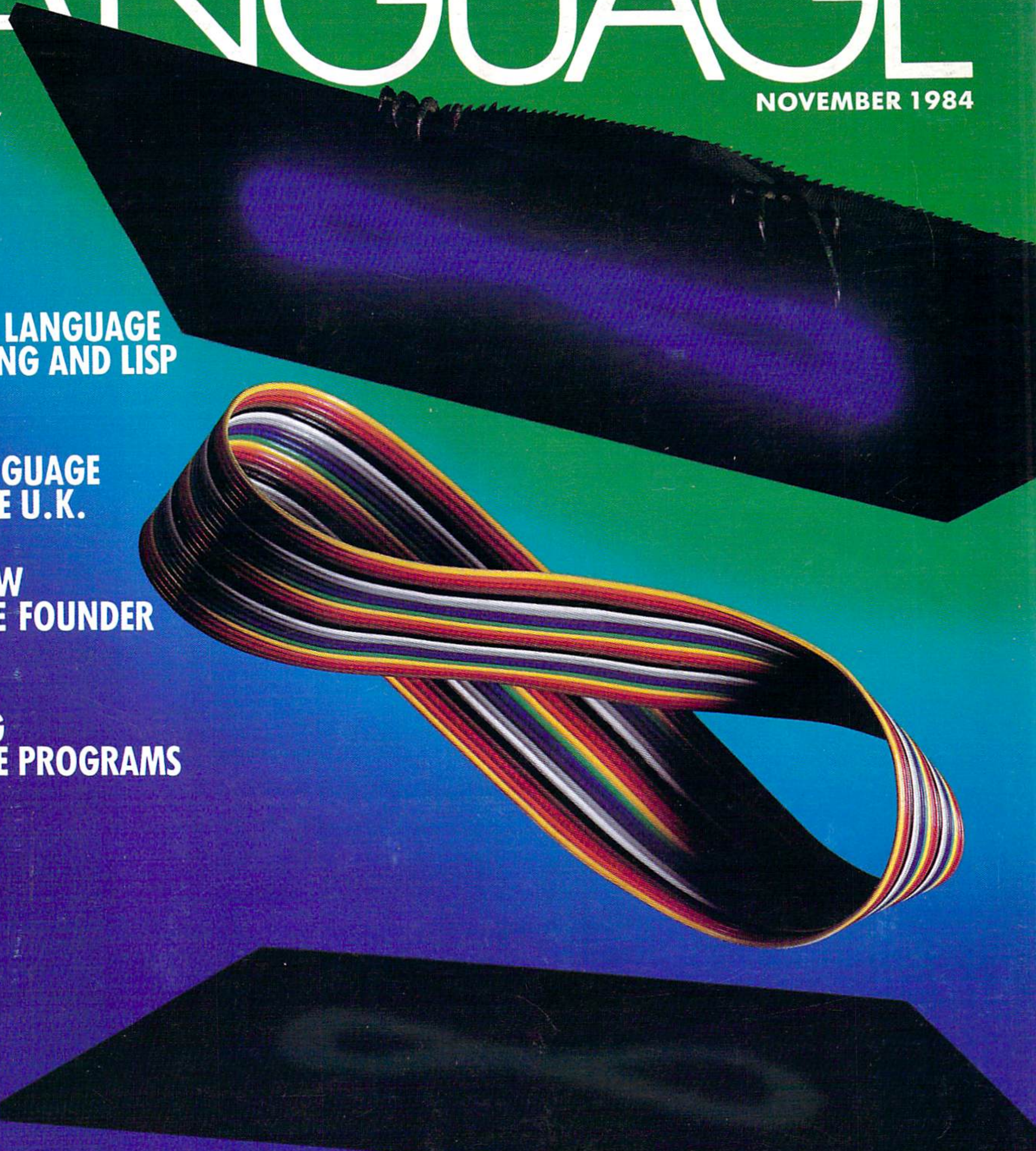
COMPUTER LANGUAGE™

$2.95

VOLUME 1,
NUMBER 3

NOVEMBER 1984



NATURAL LANGUAGE PROCESSING AND LISP

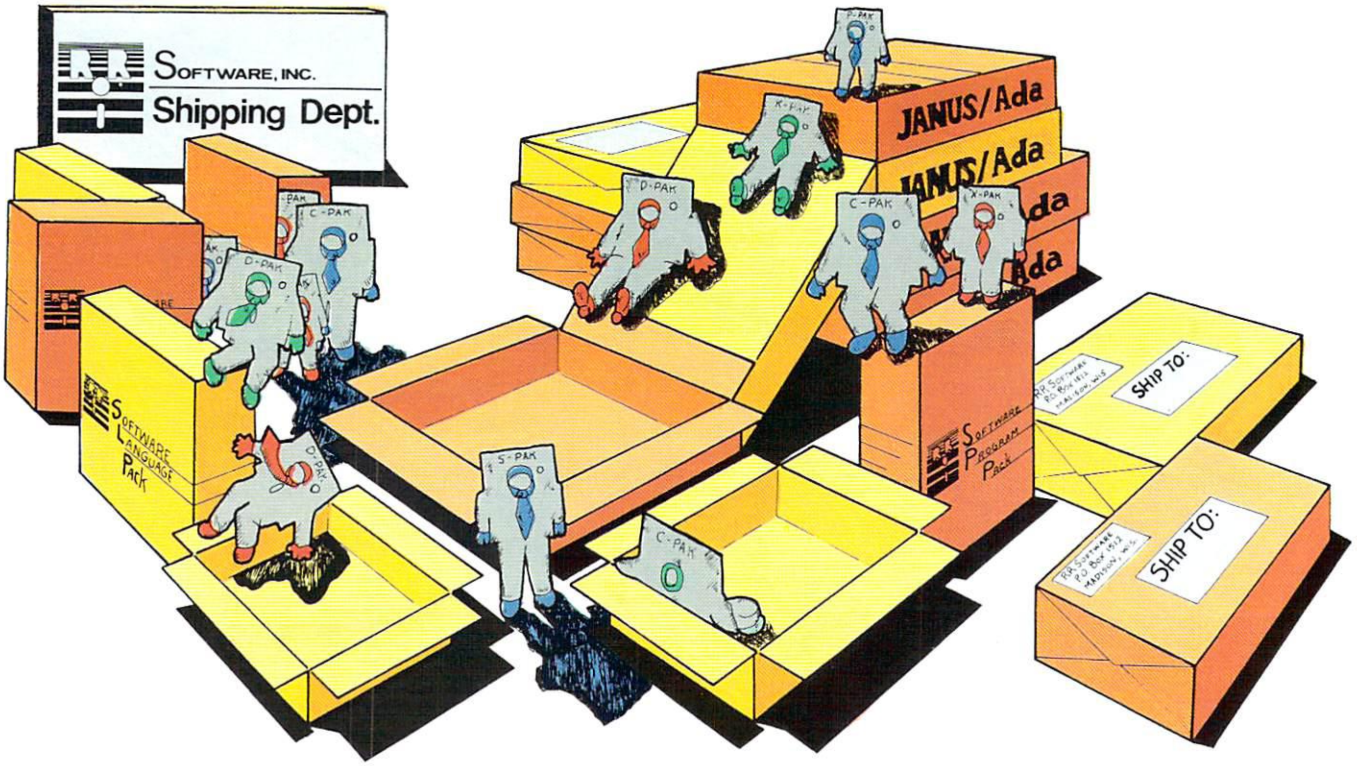
OCCAM:
NEW LANGUAGE FROM THE U.K.

INTERVIEW WITH THE FOUNDER OF CP/M

BUILDING PORTABLE PROGRAMS

ENHANCING SOURCE CODE CONTROL UNDER UNIX

WE'VE GOT YOUR PACKAGE!!



We offer you the most flexible, cost efficient means of introducing your programming staff to the Ada Language. **You** can choose the level of Support **you** need, when you need it! These Janus/Ada packages are customer-tested and available now...

- *(C-Pak) Introductory Janus/Ada Compilers*
- *(D-Pak) Intermediate Janus/Ada Systems*
- *(S-Pak) Advanced Janus/Ada Systems*
- *(P-Pak) Janus/Ada Language Translators*

- *Janus/Ada "Site" Licenses*
- *Janus/Ada Source Code Licenses*
- *Janus/Ada Cross Compilers*
- *Janus/Ada Maintenance Agreements*

Coming Soon: New Computer and Operating Systems Coverage

Selected Janus/Ada packages are available from the following:

National Distributors

Westico, Inc.
25 Van Zant St.
Norwalk, CT 06855
(203) 853-6880

Soft-Net
5177 Richard, Suite 635
Houston, TX 77056
(713) 933-1828

AOK Computers
816 Easley St., Suite 615
Silver Springs, MD 20910
(310) 588-8446

Trinity Solutions
5340 Thornwood Dr., Suite 102
San Jose, CA 95123
(408) 226-0170

Compuview Products, Inc.
1955 Pauline Blvd., Suite 200
Ann Arbor, MI 48103
(313) 996-1299

International Distributors

Micronix
11 Blackmore St.
Windsor 4030
QLD. Australia
(07) 57 9152

Progesco
155, rue du Fauburg
St. Denis
75010 Paris
(1) 205-39-47

Lifeboat of Japan
S- 13-14, Shiba
Minato-Ku
Tokyo 108 Japan
03-456-4101

CP/M, CP/M-86, CCP/M-86 are trademarks of Digital Research, Inc.
*ADA is a trademark of the U.S. Department of Defense
MS-DOS is a trademark of Microsoft
©Copyright 1983 RR Software

SOFTWARE, INC.

specialists in state of the art programming

P.O. Box 1512 Madison, Wisconsin 53701
(608) 244-6436 TELEX 4998168

CIRCLE 58 ON READER SERVICE CARD

dB COMPILER™

The first compiler for dBASE II®

SPEED

dB Compiler™ produces applications which execute substantially faster than under dBASE II® in 16-bit environments. Some operations are even faster than under dBASE III®!

INDEPENDENCE

Buy dB Compiler™ once and compile and distribute as many applications as necessary with no additional cost. WordTech imposes no licensing fees, and a compiled application will execute without dBASE II or RunTime®.

SECURITY

Compilation is far better than encryption for protecting programming insights and procedures.

PORTABILITY

dB Compiler's™ cross-environment linkers make it easy to generate executable code for several operating systems.
For CP/M-80®, CP/M-86®, PC-DOS®, and MS-DOS®.

Suggested retail price: $750; Cross-environment linkers: $350.
Corporate/Multi-user licenses available.

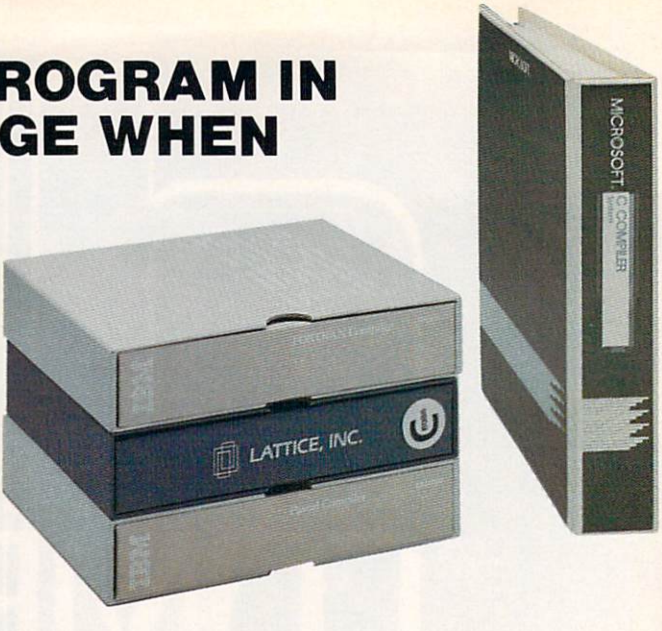
dB COMPILER™

WORDTECH SYSTEMS, INC. P.O. Box 1747 Orinda, CA 94563 (415) 254-0900

CP/M-80, CP/M-86®, DRI PC-DOS®, IBM MS-DOS®, Micro-Soft Corp., dBASE II, RunTime, dBASE III®, Ashton-Tate, Inc.

CIRCLE 41 ON READER SERVICE CARD

WHY DEBUG YOUR PROGRAM IN ASSEMBLY LANGUAGE WHEN YOU WROTE IT IN ONE OF THESE...



ATRON Announces Source Level Software Debugging

Without source level debugging, the programmer must spend time mentally making translations between assembly language and the C, PASCAL, or FORTRAN source code in which the program was written. These tedious translations burn up valuable time which should be spent making critical product schedules. The low level hex and symbolic debuggers available today are superceded by ATRON'S solution — Source Probe.

HOW TO SINGLE STEP YOUR SOURCE CODE AND KEEP CRITICAL DATA IN VIEW

With Source Probe, you can step your program by source code statements. While stepping, a window which you define can display critical high level data structures in your program. The next several source code statements are also displayed to give you a preview of what the program will do

HOW TO DISPLAY DATA IN MEANINGFUL FORMATS

Why look at program data in hex when you defined it to be another data type in your program. Source Probe provides a formated print statement to make the display of your variables look like something you would recognize. You can specify data symbolically too.

FIND A BUG — FIX IT RIGHT NOW

Source Probe provides an on-line text editor to allow you to log program corrections as you find them while debugging. With on-line display and editing of source files, the time lost printing and looking through program listings can be eliminated.

atron
SOURCE PROBE

A SNAP SHOT OF REAL TIME PROGRAM EXECUTION — BY SOURCE CODE !

When Source Probe is running on ATRON'S PC PROBE hardware, the real time execution of the program is saved. You can then view your source code as it executed in real time — including all the changes the program made to your data variables.

HOW TO FIND A BUG WHICH OVERWRITES MEMORY

When running on PC PROBE, the Source Probe can trap a bug which overwrites a memory location. Because complex pointers are normally used in high level language programming, this bug occurs frequently and is very difficult to find.

A BULLET PROOF DEBUGGER

What good is a debugger that can be wiped out by an undebugged program? With Source Probe running on PC PROBE, the software is write protected and cannot be changed.

ATRON PROVIDES THE DEBUGGING TOOLS WHICH FIT YOUR PROBLEM

PC PROBE —	A hardware aid to symbolic software debugging
SOFTWARE PROBE —	A symbolic debugger, runs without PC PROBE
SOURCE PROBE —	A source level debugger, versions run with or without PC PROBE
PERFORMANCE AND TIMING ANALYZER —	For finding where your program spends its time

WE HAVE HUNDREDS OF HAPPY CUSTOMERS

ATRON produced the first symbolic debugger for the PC and the first hardware aided debugging tool — PC PROBE. We have hundreds of happy customers who have made their schedules because of ATRON debugging tools. Why waste more time — call us today!

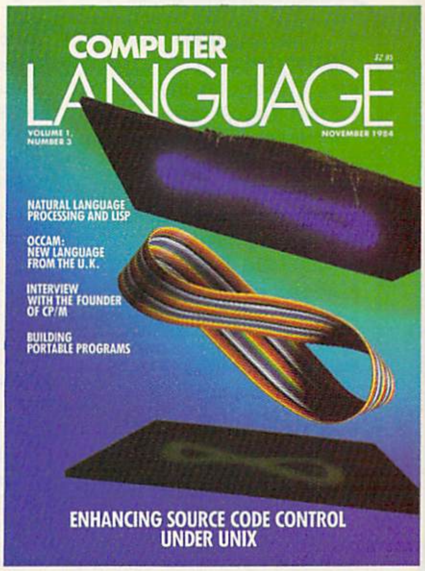
atron
a debugging company

20665 FOURTH STREET • SARATOGA, CA 95070 • (408) 741-5900

CIRCLE 4 ON READER SERVICE CARD

COMPUTER LANGUAGE

ARTICLES


COMPUTER LANGUAGE
$2.95
VOLUME 1, NUMBER 3
NOVEMBER 1984
NATURAL LANGUAGE PROCESSING AND LISP
OCCAM: NEW LANGUAGE FROM THE U.K.
INTERVIEW WITH THE FOUNDER OF CP/M
BUILDING PORTABLE PROGRAMS
ENHANCING SOURCE CODE CONTROL UNDER UNIX

Enhancing Source Code Control under UNIX **25**
by Luke C. Dion and Alan Filipski
Organizing source files in a UNIX environment is not always easy. The SCCS utilities currently provided with the UNIX operating system do not solve certain problems posed when many people are trying to store and access their files simultaneously. These authors present an enhanced version of the SCCS utility set.

Natural Language Processing and LISP **28**
by Richard Berman
The use of lists is vital to all current artificial intelligence research. Lists enable dynamic manipulation of information and are often used to simulate more structured information. Using examples of natural language processing, this author explores the power of lists, the representation of data, and recursion in LISP.

Building Portable Programs **35**
by Mark Grand
Many complex issues arise when trying to write programs that can be moved from a particular hardware and operating system environment to another. Here the author addresses some of the fundamental hurdles that must be overcome to write portably.

The Evolution of ZCPR, Part II **43**
by Richard Conn
The founder of ZCPR (Z80 Command Processor Replacement), Richard Conn, picks up his discussion from last month by addressing the issues of ZCPR3's enhanced toolset and shells.

Learn to Think in Ada **47**
by Do-While Jones
Before the power of a well-written Ada program can be appreciated, one must first learn to *think* in Ada. This author uses a simple problem—how to tell if a number is odd or even—to illustrate some of the advantages to using Ada.

DEPARTMENTS

Editor's Notes 5
Feedback 9
Industry Insight 13
A new column on issues and trends in the programming industry
Back to the Drawing Board 17
Designers Debate 21
Pascal: Just a teaching language?
Public Domain Software Review 51
Exotic Language of the Month Club 55
OCCAM: A powerful new parallel processing language from the U.K.
ComputerVisions 61
Gary Kildall, founder of CP/M
The Code Swap Shop 65
Software Reviews 66
DR FORTRAN-77, mbp COBOL, and Echelon's ZCPR3
Advertiser Index 80

Six Times Faster!

Super Fast Z80 Assembly Language Development Package

Z80ASM

- Complete Zilog Mnemonic set
- Full Macro facility
- Plain English error messages
- One or two pass operation
- Over 6000 lines/minute
- Supports nested INCLUDE files
- Allows external bytes, words, and expressions (EXT1 * EXT2)
- Labels significant to 16 characters even on externals (SLR Format Only)
- Integral cross-reference
- Upper/lower case optionally significant
- Conditional assembly
- Assemble code for execution at another address (PHASE & DEPHASE)
- Generates COM, HEX, or REL files
- COM files may start at other than 100H
- REL files may be in Microsoft format or SLR format
- Separate PROG, DATA & COMMON address spaces
- Accepts symbol definitions from the console
- Flexible listing facility includes TIME and DATE in listing (CP/M Plus Only)

SLRNK

- Links any combination of SLR format and Microsoft format REL files
- One or two pass operation allows output files up to 64K
- Generates HEX or COM files
- User may specify PROG, DATA, and COMMON loading addresses
- COM may start at other than 100H
- HEX files do not fill empty address space.
- Generate inter-module cross-reference and load map
- Save symbol table to disk in REL format for use in overlay generation
- Declare entry points from console
- The FASTEST Microsoft Compatible Linker available

SPEED! SPEED! SPEED!

- Complete Package Includes: Z80ASM, SLRNK, SLRIB - Librarian and Manual for just $199.99. Manual only, $30.
- Most formats available for Z80 CP/M, CDOS, & TURBODOS
- Terms: add $3 shipping US, others $7. PA add 6% sales tax

For more information or to order, call:
1-800-833-3061
In PA, (412) 282-0864
Or write: SLR SYSTEMS
1622 North Main Street, Butler, Pennsylvania 16001

CIRCLE 59 ON READER SERVICE CARD

S L R Systems

UNIX TIMESHARE+

**UNIX System III POWER and sophistication are yours. Let THE SOLUTION turn your micro into all you dreamed it could be, bringing the Ultimate programming environment as close as your modem. Just a local call from over 300 cities nationwide via Telenet.*

THE SOLUTION™

- **EXPANSIVE SOFTWARE DEVELOPMENT FACILITIES** including Language and Operating System design.
- **LANGUAGES:** C, Fortran 77, RATFOR, COBOL, SNOBOL, BS, Assembler + Artificial Intelligence programming via LISP.
- **USENET Bulletin Board System**—800 + international UNIX sites feeding over 190 categories, typically bringing you more than 160 new articles per day.
- **Interuser and Intersystem mail** + 'chat' capability.
- **UNIFY:** Sophisticated data-base management system.
- **UNIX & System enhancements** from U.C. Berkeley and Korsmeyer Electronic Design Inc.
- **Online UNIX manuals** + Expert consultation available.
- **SOLUTION-MART:** Hardware/Software discount shopping database.
- **LOW COST and FAST** response time. (as low as $8.95 hr. connect time + $.05 cpu sec. non-prime)
- **$24.95 = 1 hr. FREE system time** + SOLUTION News subscription + BYTE BOOK (Introducing The UNIX System 556 pp.).

*UNIX is a trademark of Bell Labs.

VISA MasterCard
Payment via VISA or Master Card

Korsmeyer
ELECTRONIC DESIGN, INC.

5701 Prescott Avenue
Lincoln, NE 68506-5155
402/483-2238
10a-7p Central

CIRCLE 34 ON READER SERVICE CARD

Editor's Notes

Of all the things happening in the computer magazine market today, one thing just doesn't quite make sense. Why are *we* growing so quickly while others are dropping like flies all around? A curious irony.

Our industry is going through what I call a "period of insecurity." Not only are magazines like *Microsystems* and *Microcomputing* now out of business, but large and small hardware and software companies are also hitting the skids. In the magazine world, though, it's sad to see so many historically important journals getting bought up by large publishing houses and then getting closed down when their profits are judged to be too low.

Integral to the success we've had so far, I feel, is the fact that our product performs a *specific* service for a *specific* audience. Because we started out small, we can afford to focus on editorial quality and avoid becoming just another expensive book of advertising.

In the coming months, you'll see *COMPUTER LANGUAGE* cover many of the same topics that *Microsystems* used to cover. We're not going to ignore the CP/M and S-100 communities, which seems to be the fashion these days. Our two-part series on ZCPR is evidence of that concern.

But the programming world has changed since 1976, and the popular machine to program for today is the IBM PC. Our goal is not to focus on the PC in particular, but to cover programming issues *in context* of all the different operating system and hardware environments popular today.

What makes *COMPUTER LANGUAGE* different from other system or software-specific journals is that it is composed of many audiences at the same time. You might say that our magazine fits in that huge, untouched grey zone between the academic and hobbiest journals. *COMPUTER LANGUAGE* is for the professional software author who is looking for practical solutions and creative technical ideas.

This month I'd like to introduce Bruce Lynch as our new Industry Insight columnist. A person with much experience and wherewithal in our industry, he will be writing a bi-monthly news column on the evolving issues and trends (and sometimes . . . gossip) in the programming world.

Also of interest this month is our exclusive interview with the founder of CP/M and chairman of Digital Research Inc., Gary Kildall. Our managing editor, Regina Starr Ridley, flew down to Monterey, Calif., to meet with this reknown technical figure. As you'll see, Kildall has some interesting things to say about the past, present, and future of our industry.

I'd like to propose that *COMPUTER LANGUAGE* is actually not a magazine at all. As a genuine *computer* forum, we're actually using these infernal machines to produce an electronic *COMPUTER LANGUAGE*. After only two weeks, our CompuServe Special Interest Group has become literally swamped with enthusiastic supporters. In just two days, we had 171 people sign up for our SIG! The articles, software reviews, and public domain code that is distributed there would take up at least five times the space in our printed magazine.

Check it out! Or, if you can't call in through CompuServe, call our remote BBS (415 957-9370) and get immediate access to our data base of public domain languages and programming utilities for many different operating system environments. As a computer magazine, it just makes sense to us that we should be reaching out to you with computers!

Craig LaGrow

Craig LaGrow
Editor

COMPUTER
LANGUAGE

EDITOR
Craig LaGrow

MANAGING EDITOR
Regina Starr Ridley

TECHNICAL EDITOR
John Halamka

EDITORIAL ASSISTANT
Hugh Byrne, Lorilee Biernacki

CONTRIBUTING EDITORS
Burton Bhavisyat, Tim Parker,
Anthony Skjellum, Ken Takara

INDUSTRY NEWS CONSULTANT
Bruce Lynch

ADVERTISING SALES
Jan Dente

CIRCULATION COORDINATOR
Renato Sunico

ART DIRECTOR
Jeanne Schacht

COVER PHOTO
Dow Clement Photography

PRODUCTION/ART
Anne Doering

PRODUCTION
Barbara Luck, Steve Campbell, Kyle Houbolt

TECHNICAL CONSULTANT
Addison Sims

MARKETING CONSULTANT
Steve Rank

ACCOUNTING MANAGER
Lauren Kalkstein

PUBLISHER
Carl Landau

COMPUTER LANGUAGE *is published monthly by* COMPUTER LANGUAGE *Publishing Ltd., 131 Townsend St., San Francisco, CA 94107. (415) 957-9353.*

Advertising: For information on ad rates, deadlines, and placement, contact Carl Landau or Jan Dente at (415) 957-9353, or write to: COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

Editorial: Please address all letters and inquiries to: Craig LaGrow, Editor, COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

Subscriptions: Contact COMPUTER LANGUAGE, *Subscriptions Dept., 2443 Fillmore St., Suite 346, San Francisco, CA 94115. Single copy price: $2.95. Subscription prices: $24.00 per year (U.S.); $30.00 per year (Canada and Mexico). Subscription prices for outside the U.S., Canada, and Mexico: $36.00 (surface mail), $54.00 (air mail)—U.S. currency only. Please allow six weeks for new subscription service to begin.*

Postal information: Second-class postage rate is pending at San Francisco, CA and additional mailing offices.

Reprints: Copyright 1984 by COMPUTER LANGUAGE *Publishing Ltd. All rights reserved. Reproduction of material appearing in* COMPUTER LANGUAGE *is forbidden without written permission.*

Change of address: Please allow six weeks for change of address to take effect. POSTMASTER: Send change of address (Form 3579) to COMPUTER LANGUAGE, *131 Townsend St., San Francisco, CA 94107.*

COMPUTER LANGUAGE *is a registered trademark owned by the magazine's parent company, CL Publications. All material published in* COMPUTER LANGUAGE *is copyrighted © 1984 by CL Publications, Inc. All rights reserved.*

NEW from BORLAND!

TURBO TOOLBOX & TURBO TUTOR

TURBO TOOLBOX
POWER TOOLS FOR TURBO PASCAL
TURBO PASCAL
MANUAL
VERSION 2.0
BORLAND INTERNATIONAL
Turbo Tutor
A Turbo-Pascal Tutorial
BORLAND INTERNATIONAL
TURBO TOOLBOX

"TURBO is much better than the Pascal IBM sells."
Jerry Pournelle,
Byte, July 1984

"TURBO PASCAL appears to violate the laws of thermodynamics.

You won't find a comparable price/performance package anywhere. It is simply put, the best software deal to come along in a long time. If you have the slightest interest in Pascal . . . buy it."
Bruce Webster,
Softalk IBM: March 1984

BORLAND INTERNATIONAL GIFT PACK

ONLY $99.95

A SAVINGS OF $30!

What a gift for you and your friends! The extraordinary TURBO PASCAL compiler, together with the exciting new TURBO TOOLBOX and new TURBO TUTOR. All 3 manuals with disks for $99.95.

TURBO PASCAL Version 2.0 (reg. $49.95). The now classic program development environment still includes the FREE MICROCALC SPREAD SHEET. Commented source code on disk
- Optional 8087 support available for a small additional charge

NEW! TURBO TOOLBOX (reg. $49.95). A set of three fundamental utilities that work in conjunction with TURBO PASCAL. Includes:
- TURBO-ISAM FILES USING B+ TREES. Commented source code on disk
- QUIKSORT ON DISK. Commented source code on disk
- GINST (General Installation Program)

Provides those programs written in TURBO PASCAL with a terminal installation module just like TURBO'S!
- NOW INCLUDES FREE SAMPLE DATABASE . . . right on the disk! Just compile it, and it's ready to go to work for you. It's a great example of how to use TURBO TOOLBOX and, at the same time, it's a working piece of software you can use right away!

NEW! TURBO TUTOR (reg. $29.95). Teaches step by step how to use the TURBO PASCAL development environment—an ideal introduction for basic programmers. Commented source code for all program examples on disk.

30 DAY MONEY BACK GUARANTEE These offers good through Feb. 1, 1985

For VISA and MASTERCARD order call toll free: **1-(800)-255-8008 1-(800)-742-1133**
(Lines open 24 hrs., 7 days a week) Dealer and Distributor inquiries welcome (408) 438-8400

CHOOSE ONE (please add $5.00 for handling and shipping U.S. orders)

___ All Three-Gift Pack	$ 99.95 + 5.00	**SPECIAL!**	___ Turbo Toolbox	$49.95 + 5.00
___ All Three & 8087	139.95 + 5.00	**SPECIAL!**	___ Turbo Tutor	29.95 + 5.00
___ Turbo Pascal 2.0	49.95 + 5.00		___ Turbo 8087	89.95 + 5.00

Check__________ Money Order__________ VISA__________ MasterCard__________

Card #: ______________________________ Exp. date: ______________ Shipped UPS

My system is: 8 bit ___ 16 bit ___

Operating System: CP/M 80 ___ CP/M 86 ___ MS DOS ___ PC DOS ___

Computer: ______________________ Disk Format: ______________________

Please be sure model number & format are correct.

NAME: ____

ADDRESS: ____

CITY/STATE/ZIP: ____

TELEPHONE: ____

California residents add 6% sales tax. Outside U.S.A. add $15.00 (if outside of U.S.A. payment must be by bank draft payable in the U.S. and in U.S. dollars). Sorry, no C.O.D. or Purchase Orders.

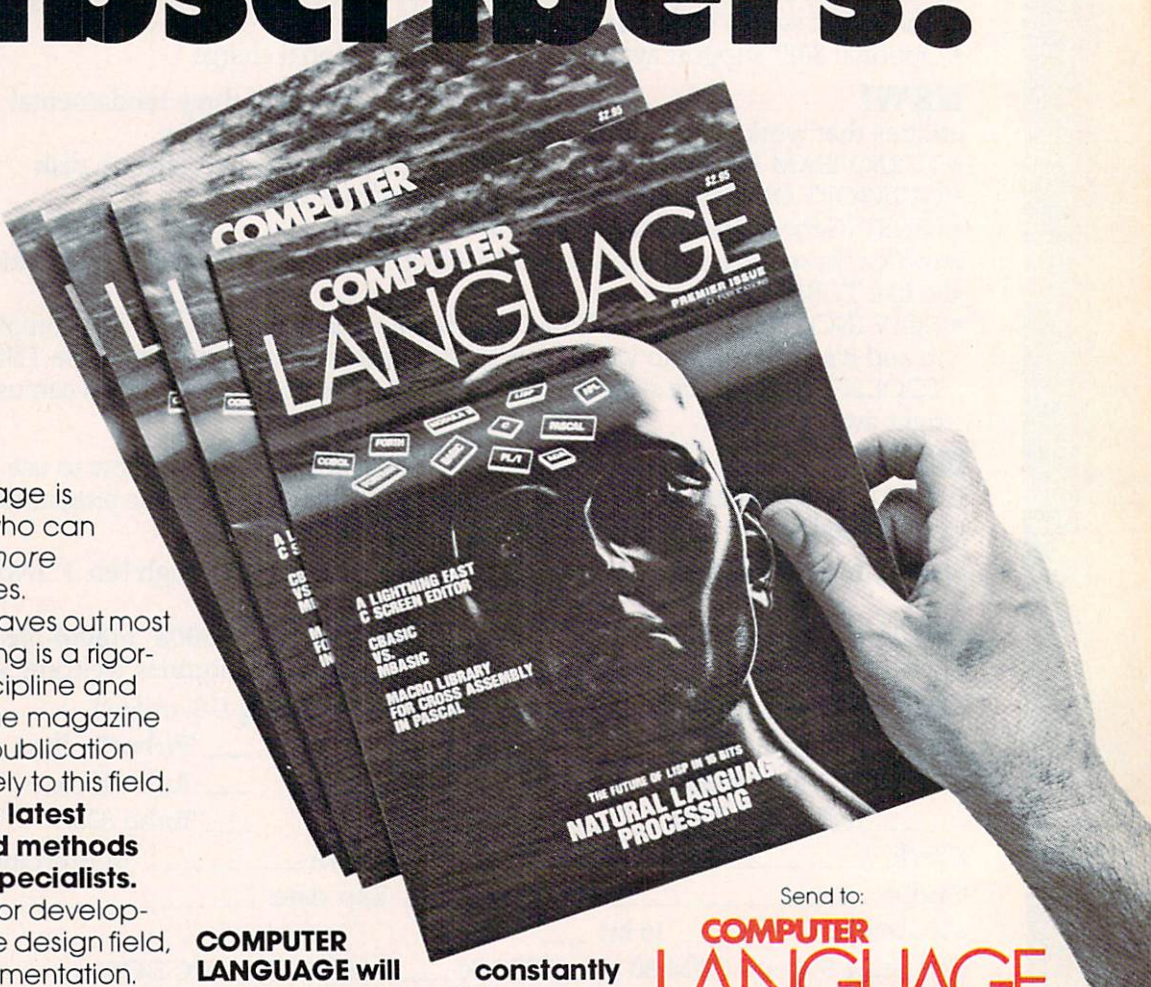
G20

BORLAND INTERNATIONAL

4113 Scotts Valley Drive
Scotts Valley, California 95066
TELEX: 172373

CIRCLE 6 ON READER SERVICE CARD

We're looking for a few good subscribers.


COMPUTER
LANGUAGE
$2.95
PREMIER ISSUE
COBOL
FORTH
COMAL
LISP
C
APL
PASCAL
BASIC
PL/I
ADA
A LIGHTNING FAST C SCREEN EDITOR
CBASIC VS. MBASIC
MACRO LIBRARY FOR CROSS ASSEMBLY IN PASCAL
THE FUTURE OF LISP IN 16 BITS
NATURAL LANGUAGE PROCESSING

Computer Language is written for people who can program in two or *more* computer languages.

Let's face it, that leaves out most people. Programming is a rigorous, intellectual discipline and Computer Language magazine is the first and only publication dedicated exclusively to this field.

Your source for the latest technical skills and methods used by software specialists.

We cover the major developments in the software design field, from theory to implementation. Computer Language focuses on the most important and useful language design information available in the fast-moving microcomputer industry.

Written for the person who takes computing seriously.

We're talking about you — the experienced software author, programmer, or engineer who routinely programs in two or more high-level languages. A person who understands the creative nature of programming and appreciates the beauty of efficient code in action.

COMPUTER LANGUAGE will constantly challenge your abilities.

The foremost industry experts will discuss: · Algorithmic Approaches to Problem Solving · Language Portability Features · Compiler Designs · Utilities · Artificial Intelligence · Editors · New Language Syntax · Telecommunications · Language Selection Criteria · Marketing Your Own Software · Critical Software & Hardware Reviews

Plus, columnists and reader forums that will put you in touch with the latest developments in the field.

Send to:

COMPUTER LANGUAGE

2443 Fillmore Street Suite #346
San Francisco, CA 94115

YES! Start my charter subscription to Computer Language. My 1 year charter subscription is just $24.00, a 33% savings under the single copy price.

☐ $24.00 Payment Enclosed ☐ Bill me.

Name

Address

City State Zip N

FEEDBACK

Turing and REXX

Dear Editor:
I read the premier issue of COMPUTER LANGUAGE and was very impressed with it. Please keep this magazine about *languages* and don't get involved in operating systems. Also try to keep it oriented toward professional rather than amateur programmers.

I would be interested in reading an article on the syntax and semantics of the programming language TURING, which was developed at a Canadian university (University of Toronto, I think). The only information I have encountered on this language was various articles in *Computerworld*, where one of the creators of the language claimed it had the power of Pascal and the ease of use of BASIC. I would like to know if these claims are true.

Another language that might be of interest to readers is the new structured interpretive language with release 3 of the VM/SP operating system. This language, called REXX, is extremely powerful and has become my favorite where I work especially since I work in a business as opposed to a systems programming environment, and the only other languages available are COBOL, RPGII, and ASSEMBLER.

I look forward to the future issues—keep up the good work!

Brook A. Everest

Forth Quicksort version

Dear Editor:

I was mystified by Richard Larson's Quicksort sample program, so here is one that I've found useful which is written in Forth. It is complete and converted to IBM/PC display management. Also, the data dependent words *COMPARE* and *EXCHANGE* are factored. The first line is the machine-dependent part and thus must be ported (debugged first). The rest of the program will run on any Forth computer (Listing 1).

Gary Nemeth
Cleveland, Ohio

ELECTRONIC
2
MAIL
2

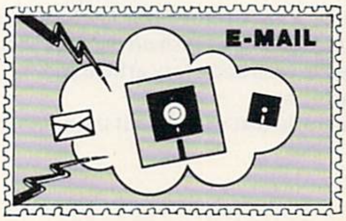


Illustration: Anne Doering

```
Standard FORTRAN Quicksort partition
C uses scratch array TEMP for temporary copy
  of segment
     Z = X(I)
     LEFT = 1
     RIGHT = J - I + 1
     DO 1 KP = I , J
          Y = X(KP)
          IF ( Y .GT. Z ) THEN
               TEMP(RIGHT) = Y
               RIGHT = RIGHT - 1
          ELSE
               TEMP(LEFT) = Y
               LEFT = LEFT + 1
          ENDIF
1    CONTINUE
     RIGHT = RIGHT + I - 1
     KT = 1
     DO 2 KP = I , J
          X(KP) = TEMP(KT)
          KT = KT + 1
2    CONTINUE
C Now X and Right are the same as Larson's

Second modification of Quicksort partition
Replace DO 1 ... 1 CONTINUE with:

for( kp=2 ; kp<=j ; kp = kp+1  ){
     y = x[kp]
     temp[left] = y
     temp[right] = y
     incleft =   y <= z
     left = incleft + left
     right = incleft - 1 + right
}
temp[right] = z
```

Listing 1.

Listings on the BBS?

Dear Editor:
While your Bulletin Board can be helpful, don't frustrate us by placing two listings in an article, then sending us to the BBS for the third! We can't always run to a terminal and modem while reading the journal.

Nicholas A. Nittler
Elkhorn, Neb.

Editor's Note: In the best of all possible worlds, we at *COMPUTER LANGUAGE* would prefer to publish *all* the code mentioned in each and every article and department.

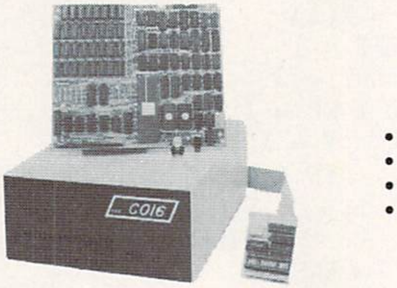
But to do that would limit the number of articles we could publish each month, and we would not be able to achieve a representative cross sampling of the evolving issues and ideas in the programming industry today. (Some program listings span over 15 to 20 pages!)

In as many cases as possible, we will continue to publish those listings that simply *must* accompany an article or department. We will also continue to maintain a remote bulletin board computer and an account of CompuServe to distribute code that is mentioned in the magazine but could not be printed for space reasons.

However, since the BBS is a long distance telephone call and many people are not subscribers to CompuServe, I would like to offer to those who wish to acquire any unpublished code to write to us, and we'll send you reprints of the code in question at no charge.

A POWERFUL 68000 DEVELOPMENT ENVIRONMENT FOR YOUR Z80 SYSTEM

CO1668 ATTACHED RESOURCE PROCESSOR



- 68000 Assembler
- C Compiler
- Forth
- Fortran 77

- Pascal
- BASIC-PLUS
- CBASIC
- APL. 68000

6 MHZ 68000 CP/M-68K 768K RAM
4 x 16081 MATH CO-PROCESSORS CPM80 RAM DISK

Develop exciting 68000 applications on your current Z80 based CPM system using powerful mini-frame like 32 bit programming languages. And then, execute them at speeds that will shame many $100K plus *mini*computer systems.

The **CO1668 ATTACHED RESOURCE PROCESSOR** offers a Z80 CPM system owner a very low cost and logical approach to 68000 development. You have already spent a small fortune on 8 bit diskette drives, terminals, printers, cards cages, power supplies, software, etc. The CO1668 will allow you to enjoy the vastly more powerful 68000 processing environment, while preserving that investment.

CO1668 ATTACHED RESOURCE PROCESSOR SPECIAL FEATURES:

- 68000 running at 6 Mhz
- 256K to 768K RAM (user partitioned between CPU and RAM Disk usage)
- Up to four 16081 math co-processors
- Real time clock, 8 level interrupt controller & proprietory I/O bus
- Available in tabletop cabinet
- Delivered w/ sources, logics, & monolithic program development software
- Easily installed on *ANY* Z80 CPM system
- CP/M-68K and DRI's *new* UNIX V7 compatible C complier (w/ floating point math) - standard feature
- *Can be used as 768K CPM80 RAM Disk*
- Optional Memory parity
- No programming or hardware design required for installation
- Optional 12 month warrantee

- PRICES START AS LOW AS $899.00 for a CO1668 with 256K RAM, CPM68K, C Complier, Sources, Prints, 200 page User Manual, Z80 Interface, and 68000 System Development Software.

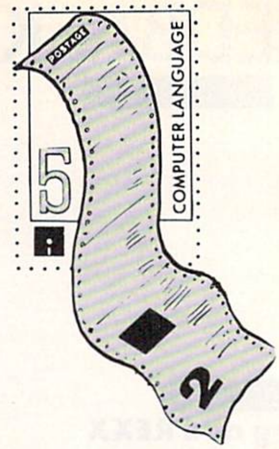
For further information about this revolutionary product or our Intel 8086 Co-Processor, please send $1 [no checks please] or call:

HSC INC.

Hallock Systems Company, Inc.
262 East Main Street
Frankfort, New York 13340
(315) 895-7426

RESELLER AND OEM INQUIRIES INVITED.

CIRCLE 31 ON READER SERVICE CARD



Amplifications on Ada

Dear Editor:
Having been involved in the Ada world for quite a while, I paid particular attention to the article by Namir Clement Shammas, "Exploring Ada and Modula-2," in the premier issue of *COMPUTER LANGUAGE*.

I offer the following corrections:

■ The declarations

```
My_Phone_Number  : String :=
      "(804)282-2294"
My_Address : String :=
      "1533 F Honey Grove"
```

are incorrect as written. When declaring a variable of an unconstrained array type, bounds must be supplied. The following represent one of many possible correct declarations:

```
My_Phone_Number  : String (1..13)
      := "(804) 282-2294";
My_Address : String (1..17) :=
      "1533F Honey Grove";
```

The bounds may be omitted when declaring a constant, as follows:

```
My_Phone_Number : constant String
      := "(805) 282-2294";
```

It is hoped the exit statement will not be used to exit nested loops. Basically, it is poor programming practice to exit more than one level at a time as this violates the single entry-single exit principle of structured programming.

■ There are five predefined exceptions in Ada: the fifth is Tasking_Error, which deals with exceptions that occur during inter-task communication.

■ The "new type" in Ada is technically called a derived type as it derives its values and operations from the parent type (the type from which it is derived).

Also I find this comparison doesn't answer the questions I would ask when first comparing two languages. The article selects small points to compare and has not followed the software engineering principle of abstraction, which allows one to focus on the most essential elements and ignore the details until a more appropriate time. As support for this statement I pose the following two questions:

■ Question 1: Why is there no mention of task types in Ada? Task types, in conjunction with access types, permit dynamic allocation of a task . . . an extremely powerful facility. Does Modula-2 have a similar capability?

■ Question 2: Why is there no mention of Ada's user-defined numeric types, which permit the designer and programmer to include in the code the constraints put on the problem by the real world? For example, in a text formatter we may want to restrict the number of characters on a line to 55. That requirement can be represented in the code by writing a user defined Integer type as follows:

```
type Character_Count_Type is
    range 1..55;
```

Any violations of that constraint will be detected at compile time or run time (depending on the kind of violation). Furthermore, a variable of type Character_Count_Type such as

```
Char_Count : Characer_Count_Type;
```

can never be mixed with a variable Line_Count of type Line_Count_Type

```
type Line_Count_Type is
    range 1..60;
Line_Count : Line_Count_Type;
```

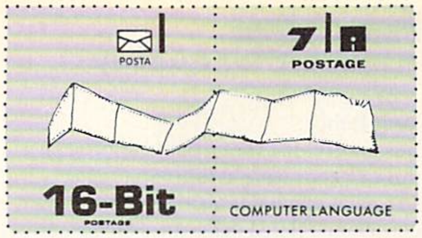
as in

```
Line_Count := Line_Count +
    Char_Count;
--**ILLEGAL**
```

The compiler will reject this assignment statement as illegal. It is illegal because variables of different types may not be mixed in one expression (unless of course type conversion is employed).

But the statement is also illogical. One doesn't normally add line count to character counts. By implementing the two features as two distinct types the logical inconsistencies are made illegal in terms of the language, and the compiler and runtime system will constantly check for these illegalities/logical inconsistencies. Does Modula-2 have a similar facility?

Putnam P. Texel
Wayside, N.J.



Be trendy and exotic!

Dear Editor:
In your premier issue Ron Jeffries mentioned three languages I've never heard of: ICON (a successor to SNOBOL, which I love), COMAL (Europe's answer to BASIC?), and Q'NIAL (which seems to be an attempt to push forward a bit). From the brief descriptions Jeffries gave, they all seem interesting. I hope you are planning articles on each of them.

I also hope you ignore the advice to "try not to get lost in the trendy issues like Ada and Forth" and "stick to to the hard-core stuff". I wonder what some people consider hard-core— FORTRAN, COBOL, and BASIC? But they were trendy once-upon-a-time, and Ada could easily be hard-core 10 or 20 years from now. Although you should not get hopelessly lost in any issue, a magazine about

GOOD NEWS!

C for the 6809 WAS NEVER BETTER!

INTROL-C/6809, Version 1.5

Introl's highly acclaimed 6809 C compilers and cross-compilers are now more powerful than ever!

We've incorporated a totally new 6809 Relocating Assembler, Linker and Loader. Initializer support has been added, leaving only bitfield-type structure members and doubles lacking from a 100% full K&R implementation. The Runtime Library has been expanded and the Library Manager is even more versatile and convenient to use. Best of all, compiled code is just as compact and fast-executing as ever - and even a bit more so! A compatible macro assembler, as well as source for the full Runtime Library, are available as extra-cost options.

Resident compilers are available under **Uniflex, Flex** and **OS9.**

Cross-compilers are available for **PDP-11/UNIX** and **IBM PC/PC DOS** hosts.

Trademarks:
Introl-C, Introl Corporation
Flex and Uniflex, Technical Systems Consultants
OS9, Microware Systems
PDP-11, Digital Equipment Corp.
UNIX, Bell Laboratories
IBM PC, International Business Machines

For further information, please call or write.

INTROL
CORPORATION
647 W. Virginia St.
Milwaukee, WI 53204
(414) 276-2937

CIRCLE 32 ON READER SERVICE CARD

computer languages must keep up with the trendy and the exotic languages.

I've been told that someone did some research on the question of what makes a good programmer and discovered that knowledge of a variety of computer languages consistently showed a strong correlation. (I'm sorry I don't remember any of the details).

Leigh Janes
Lawrenceville, N.J.

A hunger for quality

Dear Editor:
It is gratifying to see a publication such as this one appearing at this point in the stream of computer history. With so many consumer-related magazines on the newsstands, and with so many of the magazines that used to be directed to the computer professional changing their format (e.g., *InfoWorld*), we have been hurting for a quality publication to satisfy our hunger for technical interaction.

May this periodical fill this need, and prosper.

By way of suggestion, may I request some general programming articles that, instead of promoting debate on the relative merits of different languages, discuss techniques to improve the quality of the code that we write, whatever language we choose (within the constraints of each language, of course).

Again, thank you for the obvious effort and the high quality of this magazine.

William Weinman
MTR Systems Inc., Los Angeles, Calif.

COBOL rebuttal

Dear Editor:
I program on a microcomputer and I don't think COBOL is a dinosaur. Personally, I feel that COBOL has found a well-suited niche in the realm of data processing. If 60% to 70% of new application code is written in COBOL, my guess is that 60% to 70% of applications written is also in the realm of data processing. And *that's* the point which is overlooked in the article "COBOL: Pride and Prejudice."

The article overlooks two very important points. First, COBOL is functionally different from Pascal (which is functionally different from C, etc.). Second, there are applications that have conceptual frameworks that are worlds apart from each other. As an example, COBOL is a good language for accounting and payroll while C is a good language for writing operating systems. Try to write UNIX in COBOL!

A good programmer has no bias or prejudice for one language but has the ability to choose the best language for the task at hand.

The author says, "There is no theoretical or practical reason why COBOL should be slower or bigger than any other language. If anything, it should be faster." More precisely, there is no reason why it should be neither faster nor slower than any other language. The speed of a program has little if anything to do with the number of lines of code in the source language.

Whether a program performs a loop implicitly with few lines or explicitly with many, any compiler still has to generate in machine code the loop explicitly. In fact, a good compiler will recognize a loop (implicit or explicit) and generate the most optimum machine code.Voila! At the machine level, the two types of loops can not be differentiated.

I found the article's arguments to be strongly opinionated on non-substantive issues and appealing more to sensation than to technical reason. I hope that *COMPUTER LANGUAGE* will be a forum for more objective analyses in the issues forthcoming.

David Soderberg
Wethersfield, Conn.

Advertise — OR We'll
Crash YOUR DISK !
WE don't want TO GET rough
So Just go along with OUR INSTRUCTIONS.
TAKE OUT Your word processor &
Generate a 12-time insertion order to
the best Technical computer MAGAZINE
COMPUTER LANGUAGE.
ReMeMber No fUnny stUff
— OR You'll find YOUR FLOPPY
in CEMENT!

Computer LANGUAGE
131 townsend street
san Francisco, Ca 94107

415/957-9353

INDUSTRY INSIGHT

By Bruce Lynch

This bimonthly column will act as a forum and a guide for programmers. Trends in programming methods and support products will be discussed along with issues pertinent to the marketing of software and programming skills.

Your ideas and contributions are welcome. They should be sent to: Bruce Lynch, *COMPUTER LANGUAGE*, 131 Townsend St., San Francisco, Calif. 94107.

Though trends and implications of trends in programming methods and software will be covered in each column, other topics to include could range widely. Your feedback will determine what is covered. Here are some possibilities:

- The impact of non-procedural languages
- Parallel processors and the related software challenges
- Program generators—their status and future impact
- When should users program and how?
- Forecasting demand and pricing a software product
- Developing ideas for totally new rather than me-too software
- Estimating time and cost for large software projects

The microcomputer software business continues to change and mature. Though specific events cannot be accurately forecast, strong guesses can be made. They should help you be more productive and keep you from making substantial mistakes. Trends and principles can make clear what software development methods to consider and what software will be needed—how to increase the chance that efforts invested now will be of value down the road.

People have been wondering when the shakeout will begin in the microcomputer software industry. It began some time ago. It affects not just commercial software organizations but also end users, consultants, and many others.

Companies don't need to go bankrupt for a watchful observer to see a shakeout. Consider what has happened at VisiCorp, Sorcim, Digital Research, IUS, Perfect Software, Select, and many other companies that had a fair degree of leadership in this industry not long ago.

- Sorcim and IUS have both been acquired. Neither of them has a product that dominates a category.
- MicroPro's grip on the word processing market is slipping while they continue to succeed at marketing a strong contender in another category.
- VisiCorp has been battered badly by Lotus, Microsoft, and by their development philosophy.
- Perfect Software sunk huge efforts into slick advertising and marketing material and into OEM sales efforts while not paying clear enough attention to feedback from users.

Meanwhile, new successes continue, like Spinnaker and MicroRim. Most new companies that have introduced integrated software packages seem to miss the importance of several issues fundamental to their success, such as program execution speed, consistency of user interface, and clear, differentiated market positioning.

Numerous companies are trying to launch products without what I call a *critical mass*, which consists of technical quality of documentation and software, scope of marketing effort, market positioning, money, major account sales efforts, manufacturing and quality control, and a healthy combination of key company employees. The inability to meet a required threshold in any of these categories can cause a fatal flaw.

The money needed to introduce a broad-market business productivity product has gone from $1 million to $6 million in only 1 year. Expectations related to real and also perceived quality and value have risen. The size of almost all aspects of critical mass has risen—for almost any software product category. Not just software publishers are having trouble.

- Softwarebanc and Discount Software have both withdrawn from the discount mail order software business.
- Software Wholesalers, which was one of the top distributors, has fallen.
- At the same time companies like ITM seem to be doing very well.

Marketing microcomputer software is complicated. It requires careful attention by publishers to all aspects of pricing—not just the suggested retail price. Multiuser, local area network, and site-license pricing have few norms. Reseller pricing is not yet matched consistently with the values being provided by participants in the distribution channels.

Where is the magic? It is in understanding the trends and developing a critical mass.

Now for some trends and rumors in the microcomputer industry . . .

Artificial intelligence characteristics are showing up in products.

Local area networks and major account sales efforts will be fundamental to the success of numerous product categories within six months.

Copy protection is getting increased attention. The commercial software industry will come up with a combined stance soon. It is a very messy set of issues.

Development tools are becoming more flexible and make it possible to get more done in the same amount of time. Screen generators, ISAM packages, and other support software products are becoming more plentiful, more capable and lower in cost. Compilers will become substantially faster, code will be tighter, debuggers will be integrated, editing will be integrated, interpreters will be full and compatible . . .

Hardware improvements will support the improved development tools. Speedup boards for the IBM PC should become readily available within the next two or three months. Hardware support for debugging will be what it should be in the next several months.

Standards for integration, user interface, and documentation will begin to become de facto within a year.

At this point, the IBM PC AT seems like a nice machine for programming. It will accelerate existing trends in microcomputer software and will make possible software in a new class. Assume that it will soon have the following characteristics: multitasking, high-res graphics, local area network support, broader multi-user support, compilers to support virtual memory and automatic segmentation.

Look for opportunities for tightly coupled user-to-user communication and cooperation, sophisticated user interfaces, shared data bases, low-cost CAD, almost-viable expert system support, natural language applications with commercial scope, voice input, proper interfaces with videodisks and . . .

Laser printers and high-resolution, low-cost color graphics will have a serious impact on software by next summer.

By 1990, for $3,000 (1984 value) a user will purchase a machine with 4MB of RAM, 1,024 x 1,024 res color graphics, 600 meg of read-only storage, and interfaces for local area networks, printers, etc. Reasonable cost add-ons will include: videodisk with 1 gigabyte, three dimensional memory, removable storage with 100MB, typesetting quality printers, fast and hard-copy color printing, wide vocabulary voice output, tone input, voice input, touch input.

Parallelism to a high degree (like 10 or more processors cooperating together) will receive aggressive experimentation in 1985 and will be incorporated in commercial products by 1987. Multiprocessor intercommunication at memory-to-memory speeds will be a part of this. Microcomputers will have parallelism (10 to 20 CPUs) available in them for specialized low-cost applications by 1990. Nationwide telecommunications at 50K will be practical for the typical business user in 1990.

The 286 looks like it will continue to have production and performance problems. (The impact will probably hit smaller companies that want 286s.) The 32032 looks appealing but will the computer manufacturers trust National Semi? A souped-up 186 looks like a reasonable candidate for use when competing with the PC AT. Becoming available for the 8088/188 are 22-bit addressing, address translation and other enhancements.

The window still seems open for a company to develop an operating system to displace UNIX as the 2 to 5 user and local area network OS, but if it ignores critical mass the company will get slaughtered. IBM is entering that window and certainly understands the concepts behind critical mass.

IBM has been making major investments in the development of several operating systems which are targeted on microcomputers. The operating environment for the 3270 PC is the first commercial result of that effort.

The characteristics of the operating systems being developed are quite sophisticated. Rumors indicate that at least one of them will include local area networking software within the OS; one will include support for parallel processors. Support for UNIX and MS-DOS applications should also be assumed. Any company developing operating systems or making changes to existing operating systems should tread with great care—do not "bet your company" on UNIX.

The 186/188 seems to hold more promise than indicated by the negative press it has received lately. An outside manufacturer for Intel has invested substantial effort with success to create a souped-up

Announcing a

TOTAL PARSER GENERATOR

<GOAL> :: = <RAPID> <COMPILER> <DESIGN>

SLICE YOUR COMPILER DEVELOPMENT TIME

An LR(1) parser generator and several sample compilers, all in Pascal for your microcomputer.

- Generates parser, lexical analyzer and skeleton semantics
- Universal, state-of-the art error recovery system
- Adaptable to other languages
- Interactive debugging support
- Thorough documentation
- TURBO PASCAL™ INCLUDED FREE OF CHARGE
- Includes mini-Pascal compiler, assembler, simulator in SOURCE

SPECIAL INTRODUCTORY OFFER $1995

QPARSER™ runs on IBM PC/DOS in Turbo Pascal. Parser generator in object form; all else in source. QPARSER takes a grammar and generates a correct, complete, high-performance compiler with skeleton semantics in Pascal source. Easy to add full semantics for YOUR application. Excellent for industrial and academic use. An accompanying textbook (SRA publishers) available in 1985. Training can be arranged.

Educational and quantity discounts available. Check, money order, Mastercard, Visa. California residents add 6.5% sales tax.

WRITE OR CALL FOR FREE BROCHURE.
Technical details: call 408/255-5574. Immediate delivery. CALL TODAY!

QCAD
SYSTEMS, INC.
1164 Hyde Ave., San Jose, CA 95129
TOLL FREE: 800-538-9787
(California residents call 408/255-5574)

™ Turbo Pascal is a registered trademark of Borland International.

CIRCLE 23 ON READER SERVICE CARD

PRESENTING
THE
MEGAMAX C COMPILER

NEW
FOR THE
MACINTOSH

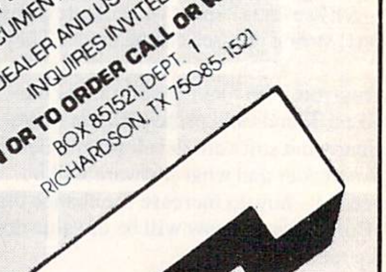
FEATURING:
• IN-LINE ASSEMBLY • ONE PASS
COMPILATION • SUPPORT OF DYNAMIC
OVERLAYS • FULL ACCESS OF MACINTOSH
TOOLBOX ROUTINES • AND MUCH MORE . . .

DEVELOPMENT SYSTEM PACKAGE INCLUDES:
• FULL-SCALE IMPLEMENTATION (K&R) C
COMPILER • THE STANDARD C LIBRARY • ROM
ROUTINES LIBRARY • LINKER • LIBRARIAN AND
DOCUMENTATION . . .

DEALER AND USER GROUP
INQUIRES INVITED

$299.95

FOR MORE INFORMATION OR TO ORDER CALL OR WRITE:
Megamax, Inc
BOX 851521, DEPT. 1
RICHARDSON, TX 75085-1521
(214) 987-4931



MACINTOSH IS A
REGISTERED TRADEMARK
OF APPLE COMPUTER INC.

CIRCLE 13 ON READER SERVICE CARD

version. It seems to have resulted in an implementation that on balance delivers more power than a similar 286. Virtual memory will be a substantial issue for some applications, but a souped-up 186 without virtual memory should be quite viable for a broad base of single user microcomputer applications. And virtual memory on a 286 is slow.

COMPAQ and AT&T have both recently announced PCs based on 8 Mhz 8086s. Though it seems a little late to be basing a new system on an 8086 rather than a 186 or 286, it seems likely that both entries will sell fairly well.

COMPAQ has about one-third of the market share for PC compatibles and seems to have a very thorough and accurate reading on the needs and reactions of the marketplace. COMPAQ's use of a green light to indicate the incremental speed when the 8086 is running at full power is an example of the company's marketing creativity. COMPAQ's careful attention to compatibility will also be a major factor in the likely success of its product line.

The AT&T micro has an interesting feature called context switching, which allows suspending of one task in order to activate another and the ability to switch back and forth. AT&T's advertising is so comprehensive that media awareness should be strong.

The HP 110 portable (and similar machines from Sharp and others) looks like a machine that could start the trend some wildly optimistic market research firms have been projecting. Think about what applications people might want to use almost any place they go or on a whimsical but high-value basis.

A product that is "coming soon" means that development work calls for the product to be marketable within six months. Such efforts, however, may never really be completed.

Language-to-language translators are being developed to convert from each of the following languages to the C language: BASIC, FORTRAN, and Pascal. Another product should be commercially available soon to accept dBase II programs as input and produce C programs as output.

Optimizing .COM and .EXE files using a disassembler combined with generalized routines for "peep hole" and other algorithms is the focus of other developments. Though the developers have experience with similar projects, such aggressive work is subject to a lot of difficult issues.

Practically every producer of high-quality C compilers for microcomputers has an effort underway to develop a source level debugger. (CWare and Williams have source debuggers available now.)

Developing libraries of C programs is popular. Intriguing projects include:

- A natural language front end that is generalized
- An expert system support program to maintain facts and rules
- Matrix manipulation routines.

Libraries with interfaces for FORTRAN, Pascal, BASIC, and C are being developed in each of the standard application support categories like screen generation, file management, and graphics. The tendency is for new entries to make products available with source and without run-time royalties. GraphiC by Scientific Endeavors, CIndex+ by TRIO

SUPER FORTH 64®
TOTAL CONTROL OVER YOUR COMMODORE-64™
USING ONLY *WORDS*
MAKING PROGRAMMING FAST, FUN AND EASY!
MORE THAN JUST A LANGUAGE...
A complete, fully-integrated program development system.
Home Use, Fast Games, Graphics, Data Acquisition, Business
Real Time Process Control, Communications, Robotics, Scientific, Artificial Intelligence

A Powerful Superset of MVPFORTH/FORTH 79 + Ext. for the beginner or professional

- 20 to 600 x faster than Basic
- 1/4 x the programming time
- Easy full control of all sound, hi res. graphics, color, sprite, plotting line & circle
- Controllable SPLIT-SCREEN Display
- Includes interactive interpreter & compiler
- Forth virtual memory
- Full cursor Screen Editor
- Provision for application program distribution without licensing
- FORTH equivalent Kernal Routines
- Conditional Macro Assembler
- Meets all Forth 79 standards+
- Source screens provided
- Compatible with the book "Starting Forth" by Leo Brodie
- Access to all I/O ports RS232, IEEE, including memory & interrupts
- ROMABLE code generator
- MUSIC-EDITOR
- SPRITE-EDITOR
- Access all C-64 peripherals including 4040 drive
- Single disk drive backup utility
- Disk & Cassette based. Disk included
- Full disk usage — 680 Sectors
- Supports all Commodore file types and Forth Virtual disk
- Access to 20K RAM underneath ROM areas
- Vectored kernal words
- TRACE facility
- DECOMPILER facility
- Full String Handling
- ASCII error messages
- FLOATING POINT MATH SIN/COS & SQRT
- Conversational user defined Commands
- Tutorial examples provided, in extensive manual
- INTERRUPT routines provide easy control of hardware timers, alarms and devices
- USER Support

SUPER FORTH 64® *is more powerful than most other computer languages!*

SUPER FORTH 64® *compiled code becomes more compact than even assembly code!*

A SUPERIOR PRODUCT
in every way! At a low
price of only
$96

Call:
(415) 651-3160
PARSEC RESEARCH
Drawer 1776, Fremont, CA 94538

Take this ad to your local dealer, or B. Dalton Bookstore. Phone orders also accepted. Immediate delivery! Dealer inquiries invited. CA residents must include tax.

or C.O.D.

® PARSEC RESEARCH (Established 1976)
Commodore 64 & VIC-20 TM of Commodore

CIRCLE 49 ON READER SERVICE CARD

SMALL C FOR IBM-PC
Small-C Compiler Version
2.1 for PC-DOS/MS-DOS
Source Code included
for Compiler & Library
New 8086 optimizations
Rich I/O & Standard Library
$40

CBUG SOURCE LEVEL DEBUGGER FOR SMALL C
Break, Trace, and Change
variables all on the
source level
Source code included
$40

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

ASM or MASM is required with compiler.
Include disk size (160k/320k), and DOS version with order.
VISA & MasterCard accepted. Include card no. & expiration date.
Washington state residents include 7.9% sales tax.
IBM-PC & PC-DOS are trademarks of International Business Machines
MS-DOS is a trademark of Microsoft Corporation.

CIRCLE 19 ON READER SERVICE CARD

Systems, CBtree by Faircom, Windows for C by Creative Solutions, and CView by CompuCraft are all examples of this trend.

New implementations of LISP, Prolog, C, Logo, BASIC, FORTRAN are all in the pipeline for MS-DOS.

By year-end the Macintosh should be supported by several more C compilers, a Modula-2, and more than one new Pascal and Forth.

Several SORT packages and libraries are being developed for integration with practically every language under MS-DOS. More than one company is attempting to develop a true "programmer's apprentice." Such a system would function as if it were an interpreter with intelligence. The intelligence would allow program fragments to be incorporated by indirect reference. Productivity gains possible through such a system could be substantial. However, such systems are highly experimental. By early 1985 a system worth experimenting with should be available from one company or another.

A low-cost C compiler that features extremely fast compile times should be available by late 1984 for use with MS-DOS. By early 1985 a Macintosh look-alike package should be available to hardware manufacturers. It is likely to include a full implementation of both the operating system and the software in the Macintosh ROM.

Assembler-to-assembler conversion between unrelated processors is a messy issue that continues to receive attention. Creating something maintainable in the target environment is a tough challenge.

Additional program execution profilers should be introduced in the next several months. Some of them should be viable for tight and easy control at the source language level for C and Pascal.

Authors are busy adding 286 instructions and MS-DOS 3.0 support to their compilers and libraries because of the PC AT. Syntax-sensitive editors are becoming more common. The issues involved are messy. ES/P by Bellesoft and FirstTime by Spruce Technology are both available for Pascal or C. PMATE by Phoenix and BRIEF by Solution Systems are being used by programmers to pick and choose those syntax-oriented features they want and to tailor them. There is a contest with a $1,000 prize to add more of such macros to BRIEF.

Support products for managing software projects are improving. Object module librarians, intelligent batch file products, source code and release control management are now available and more are coming. Project estimating and specialized documentation and graphics aids are also in the pipeline.

In the premier issue of *COMPUTER LANGUAGE* we mentioned that there are currently three companies developing SNOBOL4 compilers for the IBM PC. For your reference, here are the company names and addresses: SNOBOL4, P. O. Box 441, Millwood, N.Y. 10546; SNOBOL4+, Catspaw Inc., P. O. BOX 1123, Salida, Colo. 81201; MACRO SPITBOL, Robert B. K. Dewar, 73 Fifth Ave., New York, N.Y. 10003.

The microcomputer software business is exciting. Yes, it is maturing and becoming more costly and difficult to succeed in. That just means that you have to think things through more and find the right total combination of skills. Do not let anyone fool you into thinking that you need millions to succeed.

Watch the trends and take advantage of them. Keep your eyes wide open and you will improve both your market value and the nature of how you spend your time.

Please send me your observations and suggestions. Disagree. Take a stand. Or let me know about new products. Somewhat randomly I will send a pleasant surprise to those who take the time to write.

The C Interpreter: Instant-C™

Programming in C has never been Faster.
Learning C will never be Easier.

Instant-C™ is an optimizing interpreter for the C language that can make programming in C three or more times faster than when using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use and learn as Basic. Yet *Instant-C*™ is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

Instant Editing. The full-screen editor is built into *Instant-C*™ for immediate use. You don't wait for a separate editor program to start up.

Instant Error Correction. You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only once.

Instant Execution. *Instant-C*™ uses no assembler or loader. You can execute your program as soon as you finish editing.

Instant Testing. You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

Instant Debugging. Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload using special options.

Instant Loading. Directly generates .EXE or .CMD files at your request to create stand-alone versions of your programs.

Instant Compatibility. Follows K & R standards. Comprehensive standard library provided, with source code.

Instant Satisfaction. Get more done, faster, with better results. *Instant-C*™ is available now, and works under PC-DOS*, MS-DOS*, and CP/M-86*.

Find out how *Instant-C*™ is changing the way that programming is done. *Instant-C*™ is $500. Call or write for more information.

Rational Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), PC-DOS (IBM), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

CIRCLE 56 ON READER SERVICE CARD

BACK TO THE DRAWING BOARD

Action on the BBS

By Burton Bhavisyat

Solutions! That's what you're looking for when you go Back to the Drawing Board, and that's what you're going to find in this section of *COMPUTER LANGUAGE*. This department is dedicated to helping you find what you need to know, and this month you'll see positive proof of the process in action.

Even though *COMPUTER LANGUAGE* has only been around a few months, already our Bulletin Board System is serving hundreds of readers nationwide, giving a new meaning to the term user friendly.

Some of the friendliest users in the world are communicating via the BBS. They simply toss a question up for grabs and the answers come flying back at them, practically immediately. This month we'll take a look at the activity on the BBS for the benefit of those unfortunate readers not yet participating.

To give you some of the flavor of the BBS, messages and their responses appear as seen on your computer screen.

Msg #571 posted 08/19/84 by Dickson Leung
To: ALL About: Prolog

I have been having trouble locating an interpreter for Prolog under CP/M. I have found books on the subject and they mentioned something called micro-Prolog (in fact, one of the books is on micro-Prolog). Do you know where I can get one? Also, I would like to see an article in your magazine on artificial intelligence, especially comparing LISP and Prolog. Do you think there's any chance?

Msg #573 posted 08/19/84 by Rik Berman
To: Dickson Leung About: PROLOG

Dear Dickson, Hello. I have a CP/M version of Micro Prolog ... to get a copy try: PROLOGIC, 15102 Albright, Pacific Palisades, Calif. 90272, (213) 459-2047 or World Enterprise Robot, 3463 State St. #270, Santa Barbara, Calif. 93105, (805) 632-1604. If this fails, you can try to get in touch with the manufacturers, Logic Programming Associates Ltd. They are in London, but I have no address. Good Luck.

Msg #462 posted 08/13/84 by Anso Forth
To: ALL USERS About: NOVIX Inc.'s Forth chip

We're interested in any information on the Forth chip being developed by NOVIX Inc. and Charles Moore. Can anybody help? Even the address in Los Gatos, Calif., would be great. Thanx

Msg #538 posted 08/16/84 by Michael Ham
To: Anso Forth About: Novix Forth chip

You might try giving a call to John Golden of Golden Associates (22 Mar Monte, La Selva, Calif. 95076, (408) 688-6724). He is one of the principals and could probably give you an up-to-date progress report.

Msg #610 posted 08/21/84 by Frank Whaley
To: ALL USERS About: PC-yacc

I am in possession of a copy of UNIX yacc source—enough to build the program in a UNIX-like (read MS-DOS 2.0) environment. I am looking for someone with enough time to massage the source and build a usable copy. I am willing to share my tools in order to facilitate the project (C compiler, assembly language version of Unix standard library, latest linkers, debuggers, etc.). Drop me a line through this system and we can discuss arrangements.

Msg #690 posted 08/27/84 by Dan Miller
To: ALL USERS About: yacc, lex, and prep for the PC

Two messages have asked for yacc, a compiler compiler. Scott Guthrey of the Austin Codeworks, 11100 Leafwood La., Austin, Texas 78750, (512) 258-0785, makes available yacc, lex and prep for the IBM PC. Price—$25 for yacc and prep, $25 for lex. I met Scott at the Austin computer fair a couple of months ago and was impressed. He seems knowledgeable and nice.

He also sources a Tiny-C, a beginner's intro to a C language interpreter with a nice manual on disk for $25, a multitasking executive, and C exception macros wc, roff and grep.

I've been enjoying Tiny-C but haven't tried my copies of the yacc or lex disks so I can't give a product review yet. I think the manuals are terse, though. Suggest purchasing the manuals through Korsmeyer, a reasonably priced UNIX timeshare service.

Msg #479 posted 08/15/84 by Daniel Efron
To: ALL USERS About: C language

Hello, I am a C, Kernigan, Ritchie, and Plauger nut. If anyone has any interesting programs or words about C, leave me a message, please. Thank you.

Msg #685 posted 08/23/84 by Frank Whaley
To: Daniel Efron About: C

In your message about the C language, you failed to note what type of computer you are using and whether or not you already have a C compiler.

I am currently using:
8080/Z80 native: Whitesmith's
8088/8086: Lattice v2.12
68000 native: UniPlus (UNIX port)
68000-Z800 cross compile: Vandata

If you are interested in some UNIX-like programs for PC/MS-DOS, how about more, cat, ls, touch, rm, make, head, and tail? You might also check out my previous message on PC-yacc.

I enjoy sharing code with other programmer types (the big thrill is to have somebody else use my stuff), but a lot of what I do is proprietary to the outfit I work for. Good luck.

Msg #143 posted 08/01/84 by Anthony Skjellum
To: ALL USERS About: PC-DOS BBS software needed

Dear ALL:

I am trying to set up a bulletin board on an IBM XT computer, but I don't know where to acquire the standard bulletin software. If anyone knows where I might get such software or to whom I should speak, please send me mail on this BBS (I log in regularly). Thanks.

Msg #228 posted 08/03/84 by Bryan Oakley
To: Tony Skjellum About: RBBS-PC software

Greetings from across the universe . . . I saw your message and wish to help you if I can. I am also in the process of creating a BBS. I just recently got my software from a bulleting board system in North Dakota. Whew! A long haul. The software seems to be really good. I don't have the address, but the BBS belongs to the FARGO users group and the number is (701) 293-5977. Give them a call. For eight bucks they will mail you a copy. The last update was the middle of July, so they keep up with the times. Good luck!

Msg #310 posted 08/09/84 by James Shields
To: Anthony Skjellum About: BBS Software for your XT

Tony—I've got a board in Seattle, Wash., called the Midnight PC. I'm trying to sell the software to it. It's written in assembly, but you need not know assembly to modify the board as it's more of an interpreter than a BBS. It takes in some text files and then runs the board based on the commands you've defined there. The price is considerably more than the $8 for the other board, but I think you'd find it worth it. If you are interested, give my board a call at (206) 367-7949. I'll put you on the list of people to be allowed in.

You'll find it a bit different as it is a "room-based" system. Besides in Seattle (where there are a half-dozen or so), you'll only find two or three other room systems around the country (to our knowledge).

Msg #386 posted 08/06/84 by Jim Sills
To: Anthony Skjellum About: PC DOS software

Lyn Long in Tulsa, Okla., operates a BBS on an IBM-PC XT computer. The number of his board is (918) 749-0714. He has a complete BBS system you can download.

Msg #693 posted 08/27/84 by Dan Miller
To: ALL USERS About: etymology question

Does anyone know where the expression "the whole nine yards" or "to go the whole nine yards" comes from? Certainly it must predate football!

Msg #747 posted 09/01/84 by Dave Hilton
To: Dan Miller About: Nine yards

Hi, Dan. This is a guess only but I'll check some of my wordgame books. The "whole nine yards" comes from cloth and carpet merchants. "Nine yards" meant the whole bolt of material. Anybody else have a better guess???

Msg #771 posted 09/02/84 by Dan Miller
To: Dave Hilton About: Nine yards

[Reply to msg #747]

Thanks for the reply. That sounds like a good explanation. Let me know if you find anything definite. I know the question isn't related to computers, but I suspect many persons interested in this board, such as yourself, like language and I could get a quick answer. Thanks again.

Msg #536 posted 08/16/84 by Rik Berman
To: Gary Zablackis About: LISP

Hello Gary. I am very willing and interested in helping anyone who wishes to implement LISP. As for LISP/Prolog interpretation, Prolog is actually a child of LISP and could be implemented in LISP (as is LOGO, SMALLTALK). I would be interested in seeing an article. On the surface, it does not sound so difficult . . . anyone can contact me. Address: 21219 Community St., Canoga Park, Calif. 91304. Or I have a computer up in E-Mail most all the time. It will accept Christensen protocols (as used by MODEM, etc.) Please send batch mode—use command MODEM SB fn. ext/ DT1-818-700-1764. Thanks.

Msg #361 posted 08/03/84 by Frank Warren
To: ALL USERS About: PC-DOS/MS-DOS

I see many of you believe it would be good to have a publication devoted to MS-DOS and PC-DOS and the issues that surround them. Such a publication is already in existence! It is the SIG/86 newsletter published by SIG/86, the International MS-DOS Users Group. They've been in business about two years now and have refined their publication pretty well. Some of the recent topics covered undocumented DOS systems calls (and why not to use them), a random number generator with source, and so on.

SIG/86 is a pretty hot publication. The main authors and programmers know their stuff, and they know it well. These are some of the most experienced and seasoned 8086/8088 people around. Membership is $18 a year. To get aboard, write: Joe Boykin, 47-4 Sheridan Dr., Shrewsbury, Mass. 01545, (617) 845-1074. Their BBS number (sounds like everyone's got one these days, doesn't it?) is (617) 842-1435 at 300 baud and (617) 842-1712 for 1200 baud. Log on as GUEST.

COMPUTER LANGUAGE gets letters, too. In fact, the majority of readers don't use the BBS (can you imagine 75,000 people trying to call one phone number?), so letters are very much encouraged.

Recently Robbie Peele, a systems programmer from Atlanta, Ga., wrote about the premier issue:

"'BASIC Becomes a Structured Language' was fascinating. I think better of the authors of BASIC knowing they have realized the shortcomings of BASIC and have done something about it. I think less of Microsoft for perpetuating the old BASIC dinosaur. Can you get me any information on the availability of a True Basic compiler for the IBM PC? If not, can you get me in contact with Kemeny and Kurtz?"

True Basic Inc. is located at 39 South Main St., Hanover, N.H. 03755. If you'd like to speak to Kemeny and Kurtz directly, you can call (603) 643-3882.

Eric Schvartz, a language translator from Toronto, Canada, wrote to tell of the word-processing program he is developing and expressed a few opinions:

". . . my general impression is that software houses have never bothered to study the products used on dedicated word-processors. And if I, with my poor small machine with a fairly primitive language and without any formal training, am able to make a workable product, what couldn't they do!

"My motto is: Death to control sequences (WordStar) and formatting environments (Perfect Writer). I can assure you, from the reactions I get when I mention that I am writing a WP program, I am not the only dissatisfied user. There is room for a lot of improvements and innovations."

Schvartz also says, "I will applaud any article on operating systems if they explain how to modify them (one can live with CP/M, but it doesn't mean one has to like it!) and on alternates, if any, to CP/M for Z80 machines."

You can get programs that make a "shell" around CP/M so that the user need never know about CP/M. These shells can be very convenient and make life a lot easier. Try contacting Echelon Inc., 101 First St., Suite 427, Los Altos, Calif. 94022. They make a product called ZCPR3 (for $39!) which could well be the answer you are seeking.

Reaction to the High Touch Expert Forum (discussed in this column last month) has been outstanding. *COMPUTER LANGUAGE* has set up a data base of volunteers and their areas of expertise. Anyone needing immediate solutions to a problem can scan the list to locate someone who may know the answer. Also, of course, friendly readers can put their name and expertise on the list if they'd like to share with others. This listing can be accessed on the BBS by calling (415) 957-9370 or on the CompuServe network for all of you with CompuServe IDs.

Who says computerization is the cause of impersonalization? Our High Touch Expert Forum is about as personal as you can get. It's very private, too. Only you and your contact know about the issue being discussed.

Please don't forget the other readers, however. Just drop a line to B. Bhavisyat on the BBS (or c/o *COMPUTER LANGUAGE*) and describe what your problem was. Then tell us how you got it answered. You surely aren't the only one in the world with that problem, so you'll be doing a great public service by sharing your knowledge. A rewarding experience for you will be a revealing experience for us.

Let's go for it! ■

Only $95 with FULL SOURCE CODE!

Q/C

"... an incredible learning tool." *Byte*

For only $95, Q/C is a ready-to-use C compiler for CP/M with complete source code. Here's what *BYTE* (May 1984) said: "Q/C ... has a portable library and produces good code quality. If you want to learn compiler construction techniques or modify the standard language, Q/C is the obvious choice."

- Source code for compiler and over 75 library functions.
- Strong support for assembly language and ROMs.
- No license fees for object code.
- Z80 version takes advantage of Z80 instructions.
- Q/C is standard. Good portability to UNIX.

Q/C has casts, typedef, sizeof, structure initialization, and function typing. It is compatible with UNIX Version 7 C, but doesn't support long integers, float, parameterized #defines, or bit fields. Call about our new products: Q/C profiler, Z80 code optimizer, and Z80 assembler and virtual linker, all with full source code!

THE CODE WORKS
5266 Hollister, Suite 224
Santa Barbara, CA 93111
(805) 683-1585

Q/C, CP/M, Z80, and UNIX are trademarks of Quality Computer Systems, Digital Research, Zilog, Inc., and Bell Laboratories respectively.

CIRCLE 42 ON READER SERVICE CARD

COMPUTER RESOURCES of WAIMEA

* * * EASY TO USE * * *
Macro Programs for

Spellbinder™

We have been using and working with Spellbinder since late 1981. We use computers extensively in the day-to-day operation of our business and have developed a number of programs which we find useful. We recently formed a software development and marketing company - Computer Resources of Waimea, to promote and market these programs, most being enhancements and macro programs running under Spell binder. Spellbinder's macro programming language M-Speak is extremely versatile and in our opinion is one of the best kept "secrets" in the world of micro computers. We have a number of macro programs for the end user, a number of utilities for the programmer, and for those who want a more or less organized instruction set for M-Speak, our head programmer has compiled his personal notes into a booklet which the M-Speak user should find very useful. It can be purchased for $10.00. Send for our complete listing.

P.O. Box 1206 Kamuela, Hawaii 96743
(808) 885-7905

CIRCLE 14 ON READER SERVICE CARD

MicroMotion

MasterFORTH

It's here — the next generation of MicroMotion Forth.

- Meets all provisions, extensions and experimental proposals of the FORTH-83 International Standard.
- Uses the host operating system file structure (APPLE DOS 3.3 & CP/M 2.x).
- Built-in micro-assembler with numeric local labels.
- A full screen editor is provided which includes 16 x 64 format, can push & pop more than one line, user definable controls, upper/lower case keyboard entry, A COPY utility moves screens within & between lines, line stack, redefinable control keys, and search & replace commands.
- Includes all file primitives described in Kernigan and Plauger's Software Tools.
- The input and output streams are fully redirectable.
- The editor, assembler and screen copy utilities are provided as relocatable object modules. They are brought into the dictionary on demand and may be released with a single command.
- Many key nucleus commands are vectored. Error handling, number parsing, keyboard translation and so on can be redefined as needed by user programs. They are automatically returned to their previous definitions when the program is forgotten.
- The string-handling package is the finest and most complete available.
- A listing of the nucleus is provided as part of the documentation.
- The language implementation exactly matches the one described in FORTH TOOLS, by Anderson & Tracy. This 200 page tutorial and reference manual is included with MasterFORTH.
- Floating Point & HIRES options available.
- Available for APPLE II/II+/IIe & CP/M 2.x users.
- MasterFORTH – $100.00. FP & HIRES – $40.00 each
- Publications
  - FORTH TOOLS – $20.00
  - 83 International Standard – $15.00
  - FORTH-83 Source Listing 6502, 8080, 8086 – $20.00 each.

Contact:

MicroMotion
12077 Wilshire Blvd., Ste. 506
Los Angeles, CA 90025
(213) 821-4340

CIRCLE 40 ON READER SERVICE CARD

Multi-Basic

"The BASIC compiler that compiles both MBASIC and CBASIC"

Now you don't have to give up the features you like about MBASIC to obtain the powerful capabilities of CBASIC. Multi-Basic gives you both.

Multi-Basic works with your existing programs so your current software investment is protected. But just as important, Multi-Basic opens the door to a whole new way of programming. With Multi-Basic you can write very readable, modular and structured programs. Multi-Basic makes program maintenance as easy as it is with Pascal.

In addition to understanding the two most popular dialects of BASIC, Multi-Basic allows you to extend the language even further. You can add your own statements and functions as needed.

Multi-Basic is also compatible with our Pascal and C compilers. This allows your BASIC programs to use routines written in Pascal or C.

In today's fast changing computer business, you need a language as versatile as Multi-Basic. Invest a little time today and save a lot of time tomorrow. You owe it to yourself to see what a difference Multi-Basic can make.

Multi-Basic is available for the TRS80 models I, II, III, 4 and 12; Tandy 2000, IBM PC, and CP/M. It is compatible with TRSDOS, LDOS, NEWDOS, DOSPLUS, MSDOS, PCDOS, CP/M and CP/M plus.

Alcor Multi-Basic $139

Other Products:

Advanced Development Package	$ 69
Blaise I Text Editor (Mod 1 or 3)	$ 49
Blaise II Text Editor (all others)	$ 79
Multiprocessor Assembler	$ 69
Alcor C	$139
Alcor Pascal (for CP/M, MSDOS, PCDOS)	$139
Complete Development System includes compiler, text editor and advanced development package	$250
Shipping U.S.A.	$6.00
Shipping Overseas	$28.00

ALCOR Systems

13534 Preston Road, Suite 365
Dallas, Texas 75240
(214) 494-1316

Multi-Basic is a trademark of Alcor Systems
TRS80 is a registered trademark of Tandy Corporation
CP/M, CBASIC are trademarks of Digital Research
MSDOS, MBASIC are trademarks of Microsoft

CIRCLE 1 ON READER SERVICE CARD

DESIGNERS DEBATE

Pascal: Just a teaching language?

By Ken Takara

Our debate this month concerns Pascal, that ubiquitous language now taught in nearly every computer science curriculum.

The word "Pascal" has become almost synonymous with the phrase "structured programming," which in turn implies *better*.

Is Pascal better? Considering the number of shops that use it, it must be good. It definitely has a most conspicuous position among the many programming languages. It is one of the first "designed" languages and one of the first to become available on microcomputers.

Even so, there seems to be a trend toward Pascal shops migrating to the C language. Why?

Pascal was designed as a language for teaching programming, language design, and compiler construction back in the early 1970s, the days of the mainframe, IBM, FORTRAN, and COBOL. It followed the path that was first laid down by ALGOL, the granddaddy of the modern, structured language world.

Pascal's wide acceptance among universities led to students graduating with a thorough foundation in its use. And, after its initial exposure in the academic world, it took a prominent position in the real world of industry.

As with most languages, various versions of Pascal are loose in the world. In this debate, we refer primarily to the ISO standard Pascal as defined by Kathleen Jensen and Niklaus Wirth in *Pascal User Manual and Report*.

The Pascal protagonist is Elbert Hinson, a senior analyst. He also teaches Pascal. On the offensive is Bruce Hunter, writer.

Let's start off with the business of Pascal's strict structuring . . .

Hunter: Actually, the ISO standard is really just a proposed standard—it hasn't been accepted yet. When Wirth came up with Pascal, what he wanted to do was introduce a structured teaching language patterned more or less after ALGOL.

Incidentally, ALGOL is an interesting language that most of our modern programming languages are patterned after but which isn't itself in much use anywhere.

Basically, Wirth knew he'd have to write the compiler himself, and he wanted to keep it as easy as possible. To do that, he wanted to reduce the amount of parsing necessary.

Hinson: The Pascal compiler is designed to be a single pass compiler, so there is a very rigid order for declarations. Not only do you have to declare everything before you use it, they have to be defined in a specific order—tables first, then constants, types, variables, and finally procedures and functions.

Hunter: This is nice for beginning programmers. But, on the other hand, when you're working with real applications programs or when you're involved in systems work you don't want to be tied to that sort of thing. You want to be able to lay them out in any order. They're necessary, but you don't want to have to bother with them.

One of the big things at the time was the concept of top-down programming. In languages like PL/I or ALGOL, you deal first with the main procedure, then the second level, and so on. Now we've got Pascal, which is totally upside down, where the most trivial things appear first. The last thing to appear is the most important part: the program name and main procedure.

The programmer has to write bottom up rather than top down, defeating one of the major points of emphasis.

Hinson: I disagree. I don't see how that affects top-down programming.

With the editors available nowadays, like WordStar, you don't have to actually write the program in that order. You can start by writing the main program first, then adding the functions ahead of it.

Another method I've used is to build a skeleton main program with stubs for the functions and subroutines.

Even with PL/I, you need to create these "dummy" routines while you check out the main procedure. And then you simply replace each dummy with the real thing as you get to it.

Anyway, you're supposed to plan the program before you code it.

Hunter: Because Pascal's a teaching language, Wirth wanted it as compact and lightweight as possible. In order to do that, he minimized the number of data types available.

He only included scalars—data structures that can hold only a single value. Consequently, niceties like strings are nonexistent.

So you have to handle strings as arrays of characters. Well, strings and arrays of characters are very different things. A string has a length associated with it and a number of specialized operations.

Try to input an array of characters as a string. For example, you normally use a carriage return to end a string input. If the carriage return is used to delimit the end of the string, then you have lost the use of that character by the program.

Hinson: That's not altogether true either. Not all data types are scalar. Pascal has the type *record*, with which you can put together complex data structures with mixed types. You can define a record with an integer followed by an array of characters to create a string, for example.

Sure, you have to create your own string input routine that terminates conditionally when it sees a carriage return, but this is hardly a problem solely with Pascal.

And Pascal does have the *set* type, which is very powerful for working with non-numeric data items. You don't have to simulate these items using integers as you would with another language.

Hunter: Yes, the strong data typing and the successor and predecessor functions allow you to work with sets exquisitely well.

I've heard that I/O and system access in Pascal are not very good, making real programming in Pascal difficult. Would you comment on this?

Hunter: Pascal I/O is very limited. This particular language cannot do random I/O, it only can handle ASCII files, and once you've closed a file it's closed forever—you can't open and append to it.

Input is difficult enough, but output is worse. For example, Wirth allowed for only right justification. This means programs that require precisely formatted output are very difficult to write.

In my book, *Fifty Pascal Programs*, I have an example of a little business program that prints out a report. I have to actually count characters and append blanks to get things justified. It gets to be extremely painful.

Hinson: I don't think it's that bad. You can specify the position for a decimal value with an instruction like *write (X:10:2)*, which gives you a 10-digit decimal value with two digits to the right of the point and very nicely aligned dollar amounts. Otherwise it will be right justified, as you usually want it. And text is always left justified, which is also as you would want it.

But then, this is a problem common to most scientific languages. If you want format, you ought to be using RPG or COBOL. The problem with files is very real, though, and represents a major drawback with standard Pascal.

Of course, the philosophy of Pascal is to avoid hardware-specific constructs, so I/O is left up to the programmer and is not really part of the language.

Hunter: The Pascal compiler Wirth designed is not really a compiler; it's called a p-code generator. It compiles the Pascal code into p-code which is then run through an interpreter.

BASIC, another teaching language, is interpreted, and this means that you can enter a few lines and try it out right away. Sure, it's slow, but that's okay.

The Pascal compiler, however, is neither here nor there. You don't have the ease of execution provided by an interpreter nor do you have the speed of a compiled language.

You don't have separate compilation in Pascal either, for a similar reason. It's just too high a level of complexity for the Wirth p-code generator.

Hinson: The p-code generator is not specified in the language design. Wirth never said how to implement Pascal. Versions of Pascal are available fully compiled to machine code.

There are also p-code versions—some of which, though not within the standard, have separate compilation. The lack of separate compilation in the standard is a problem, no doubt about it.

Without the modularity you can't build up function libraries. Of course, this is no real problem for a teaching language. I guess that's one of the reasons Wirth went on to develop Modula-2.

Can you make any comments about Pascal's programming style?

Hunter: When you write a program, you want it to be simple, clear, and straightforward. With standard Pascal, in order to do anything you have to be clever, and clever code is not easy for anyone to read or understand.

Pascal programmers have built up all sorts of tricks to get around the limitations, and many of these are very difficult to follow. You have to know the tricks yourself in order to know what the programmer is doing.

A more complete language may be more intimidating in the beginning because of its size, but the programmer tends to write clearer code because it's much easier to write. You never have to be clever, and it's far easier to read. Languages like PL/1 or Ada or C are like this. C, for example, is a very broad language, with such an extremely rich set of instructions available that you never really have to kluge the language.

Hinson: I haven't seen that problem very often. It's true that systems work is generally difficult in Pascal because it hides the computer from the programmer, resulting in the need for various tricks. But otherwise, you can create most of the functions you need in a straightforward manner. For example, Allan Miller's *Pascal Programs for Scientists and Engineers*

CP/M-80 C Programmers . . .

Save time

. . . with the BDS C Compiler. Compile, link and execute *faster* than you ever thought possible!

If you're a C language programmer whose patience is wearing thin, who wants to spend your valuable time *programming* instead of twiddling your thumbs waiting for slow compilers, who just wants to work *fast*, then it's time you programmed with the BDS C Compiler.

BDS C is designed for CP/M-80 and provides users with quick, clean software development with emphasis on systems programming.

BDS C features include:

- Ultra-fast compilation, linkage and execution that produce directly executable 8080/Z80 CP/M command files.
- A comprehensive debugger that traces program execution and interactively displays both local and external variables by name and proper type.
- Dynamic overlays that allow for run-time segmentation of programs too large to fit into memory.
- A 120-function library written in both C and assembly language with full source code.

Plus . . .

- A thorough, easy-to-read, 181-page user's manual complete with tutorials, hints, error messages and an easy-to-use index — it's the perfect manual for the beginner *and* the seasoned professional.
- An attractive selection of sample programs, including MODEM-compatible telecommunications, CP/M system utilities, games and more.
- A nationwide BDS C User's Group ($10 membership fee — application included with package) that offers a newsletter, BDS C updates and access to public domain C utilities.

Reviewers everywhere have praised BDS C for its elegant operation and optimal use of CP/M resources. Above all, BDS C has been hailed for it's remarkable *speed*.

BYTE Magazine placed BDS C ahead of all other 8080/Z80 C compilers tested for fastest object-code execution with all available speed-up options in use. In addition, BDS C's speed of compilation was almost *twice* as fast as its closet competitor (benchmark for this test was the Sieve of Eratosthenes).

"I recommend both the language and the implementation by BDS very highly."
Tim Pugh, Jr.
in *Infoworld*

"Performance: *Excellent.*
Documentation: *Excellent.*
East of Use: *Excellent.*"
InfoWorld
Software Report Card

". . . a superior buy . . ."
Van Court Hare
in *Lifelines/The Software Magazine*

Don't waste another minute on a slow language processor. Order your BDS C Compiler today!

Complete Package (two 8" SSDD disks, 181-page manual): **$150**
Free shipping on prepaid orders inside USA.
VISA/MC, COD's, rush orders accepted.
Call for information on other disk formats.

BDS C is designed for use with CP/M-80 operating systems, version 2.2. or higher. It is not currently available for CP/M-86 or MS-DOS.

BDSoftware

BD Software, Inc.
P.O. Box 2368
Cambridge, MA 02238
(617) 576-3828

CIRCLE 5 ON READER SERVICE CARD

has a collection of all sorts of mathematical and engineering algorithms in standard Pascal, and there is nothing obscure in them.

As for clever or obscure code, you can do that with any language. Look at C, where you can put together a three-line subroutine that is absolutely impossible to figure out.

Writing clean, straightforward code always requires discipline.

Hunter: Generally, when people argue Pascal, they're talking about some enhanced version. And you can't argue for the enhancements for the simple reason that there is no standard enhanced set.

If I were to write a program in Oregon Pascal, then try to port it to UCSD Pascal or Pascal 68K, it wouldn't go. So with no standards for the enhancements, you don't have a portable language.

And this brings you back to the Jensen and Wirth version which isn't going to do the job.

Hinson: An ANSI committee has been working on a standard enhanced set for Pascal. Jerry Pournelle of *Byte* had a forum on it at the last West Coast Computer Faire. Among other things, they have instituted some changes such as the ability to pass varying length arrays as parameters to functions.

That standard may have been accepted by now.

Where do you find Pascal useful?

Hunter: When a programming firm is recruiting young programmers right out of school, they can be sure that, even if they know nothing else in the world, they'll know Pascal.

They can hire inexperienced programmers, start them off in an extremely robust set of Pascal, then retrain them in Ada. For this sort of thing, Pascal can be very useful since Ada is a natural progression from Pascal.

Hinson: I like to use Pascal to experiment.

It's easy to code algorithms in Pascal to see how well they work and then convert them to FORTRAN, which I use at work. I find I can write very elegant, clean functions in Pascal. I don't have problems with a limited set of operations.

Pascal is obviously an excellent language for teaching programming style and techniques by virtue of its strong typing, simplicity, and clean and elegant structure.

However, these strengths metamorphose into severe liabilities when it is taken from its academic environment and put to duty in the real world.

The use of Pascal for systems work is contrary to its philosophy of hiding the computer from the programmer, for example. And the primitive I/O structure can prove a hindrance to many data-intensive programs. The result is many enhanced versions, all different, contradicting the objective of portability.

The rise of Modula-2 and the existence of the various structured languages like C pay mixed tribute to the strengths and weaknesses of Pascal. It appears, though, that Pascal will be around for a very long time.

The ultimate reference for Pascal is, of course, the classic, *Pascal User Manual and Report* by Kathleen Jensen and Niklaus Wirth, published by Springer-Verlag. Other books mentioned here were *Fifty Pascal Programs* by Bruce Hunter, published by Sybex, and *Pascal Programs for Scientists and Engineers* by Allan R. Miller, also published by Sybex. ■

For your IBM/PC

mbp COBOL: 4 times faster, and now with SORT & CHAIN.

$750.

mbp COBOL can be summed up in one word: fast.

Because it generates native machine language object code, the mbp COBOL Compiler executes IBM/PC* programs *at least* 4 times faster (see chart).

GIBSON MIX Benchmark Results
Calculated S-Profile
(Representative COBOL statement mix)
Execution time ratio

mbp COBOL	Level II** COBOL	R-M*** COBOL	Microsoft**** COBOL
1.00	4.08	5.98	6.18

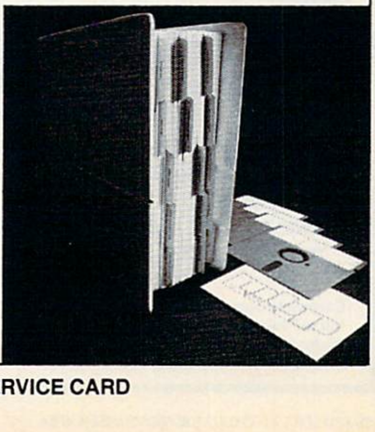
128K system with hard disk required. *IBM/PC is an IBM TM; **Level II is a Micro Focus TM; ***A Ryan-McFarland TM; ****A Microsoft TM.

Fast also describes our **new SORT**, which can sort four-thousand 128-byte records in less than 30 seconds. A callable subroutine or stand-alone, 9 SORT control fields can be specified. And our **new CHAIN** is both fast and secure, conveniently transferring control from one program to another, passing 255 parameters. Plus, **new extensions** to ACCEPT & DISPLAY verbs give better, faster interactive programming.

The complete COBOL. An Interactive Symbolic Debug Package included standard; Multi-Keyed ISAM Structure; listing options allow source & object code, map & cross-reference checking; GSA Certification to ANSI '74 Level II; mbp has it all.

It's no surprise companies like Bechtel, Chase, Citicorp, Connecticut Mutual, and Sikorsky choose mbp COBOL; make it your choice, too. mbp is available at Vanpak Software Centers, or direct. For complete information, write **mbp Software & Systems Technology, Inc.,** 7700 Edgewater Drive, Suite 360, Oakland, CA 94621, or phone 415/632-1555 —today.

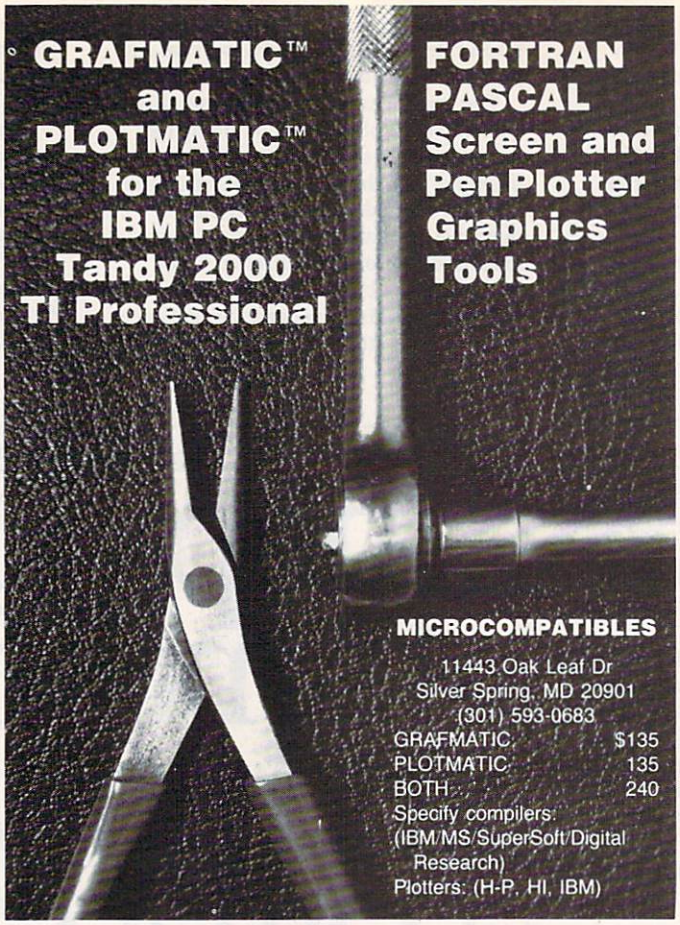
mbp®



CIRCLE 39 ON READER SERVICE CARD

Total Support Packages

The GRAFMATIC (screen graphics) and companion PLOTMATIC (pen plotter) libraries of modular scientific/engineering graphics routines let you easily create 2D and 3D plots in customized or default formats. Pen plot preview with GRAFMATIC. Plot interactively or in deferred mode. Others only provide our "primitives" (mode, color, cursor, character, pixel, line, paint . . .). We follow through with: auto-scaling, auto-axis generation, auto-tic mark labeling, function plots, tabular plots, auto-function plots (complete plot in default format with one easy call), auto-tabular plots, log/parametric/contour plots, 3D rotation/scaling/translation, wire frame model (for old time's sake), hidden line removal for solid models (GRAFMATIC only), cubic and bicubic spline interpolants, least squares fits, bar and pie charts, screen dump, . . . You name it, We have it! Best of all, the clearest and most complete documentation to be found in microcomputerland. User support? Of course, call us! We offer a no questions asked money-back guarantee.



GRAFMATIC™ and PLOTMATIC™ for the IBM PC Tandy 2000 TI Professional

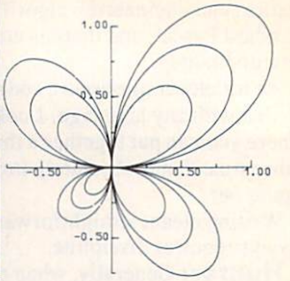
FORTRAN PASCAL Screen and Pen Plotter Graphics Tools

MICROCOMPATIBLES

11443 Oak Leaf Dr
Silver Spring, MD 20901
(301) 593-0683

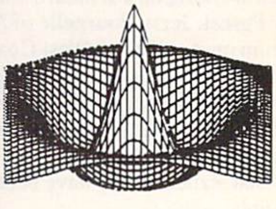
GRAFMATIC	$135
PLOTMATIC	135
BOTH	240

Specify compilers: (IBM/MS/SuperSoft/Digital Research)
Plotters: (H-P, HI, IBM)



". . . GRAFMATIC is the most imaginative and well designed use of FORTRAN I have yet seen in a FORTRAN microcomputer software package."

James Creane,
Contributing Editor,
Personal Computer Age



CIRCLE 15 ON READER SERVICE CARD

Use ALL the Power of Your MS-DOS, IBM PC-DOS, or CP/M-80 System with UNIX-Style Carousel Tools

CAROUSEL TOOL KIT
CAROUSEL TOOL KIT
CAROUSEL TOOL BOOK

```
ch "CP/M" "MS-DOS" <doc >newdoc
diff newdoc doc I more
ed newdoc
kwic newdoc I sortmrg I uniq I unrot >index
make -f makdoc ndx
```

Carousel Tools and Carousel ToolKits are trademarks of Carousel MicroTools, Inc. CP/M is a trademark of Digital Research; IBM is a trademark of International Business Machines; MS is a trademark of MicroSoft; UNIX is a trademark of Bell Laboratories.

CAROUSEL TOOLS are a proven set of over 50 programs designed to be used with pipes, redirected I/O and scripts. In the style of UNIX each Tool does one thing well, and the Tools can be used together to do more complex tasks.

YOU ACCOMPLISH MORE using Carousel Tools: better programming and documentation support, simpler data and file housekeeping, more general file handling.

TOOLS FOR PC/MS-DOS 2.x AND CP/M-80 are available now. The DOS ToolKit is $149. The CP/M ToolKit is $249 and includes a *shell* to provide pipes, redirected I/O, and scripts. Source code is available for $100 more.

ORDER YOUR TOOLKIT TODAY. VISA master charge

CALL OR WRITE:

CAROUSEL MICROTOOLS, INC.
609 Kearney Street, El Cerrito, CA 94530 (415) 528-1300

CIRCLE 8 ON READER SERVICE CARD

Enhancing Source Code Control under UNIX

PART I

By Luke C. Dion and Alan Filipski

The UNIX Source Code Control System utilities provide the individual programmer with a mechanism for the orderly updating of source files under the UNIX operating system. But for managing source files in the medium-to-large software project world, SCCS is not adequate.

It is possible, however, to provide an acceptable source file management facility for such a project by providing a front-end interface to the SCCS utilities. This article describes the SCCS utilities and the creation of such an interface (known as Project SCCS or PSCCS) for use in a UNIX operating system port project.

What is a SCCS?

Organizing a programmer's source files for easy retrieval is a not always an easy-to-solve problem. When the files are generated by an organization of programmers, the problem is far worse. Not only are there a large number of source files (for example, the UNIX operating system needs more than 5,000), but it's necessary to maintain multiple versions of the "same" file.

Work continues on a program after it is released as a product, but it is important to be able to generate the product as released—for example, to verify bug fixes sent to the field. The relationships among files must be recorded somewhere. Which source files must be compiled and linked together to produce which object files? Where is the documentation related to these source files? How can we prevent several programmers from simultaneously editing the same file?

Some sort of data base is evidently necessary. A primitive first step in this direction is the very existence of the hierarchical directory system available under the UNIX operating system.

Utility source files, for example, can all be kept under a single directory with subdirectories for specific utilities or families of utilities. This makes files quite a bit easier to find but does not come close to providing adequate structure to maintain a project manager's sanity during a task such as a UNIX operating system port.

UNIX SCCS utilities

SCCS utilities furnished by the UNIX operating system provide a next step in source file organization. In particular, they are an efficient mechanism for saving the history of changes to a source code file. This allows regeneration of any of a series of released versions of an evolving source file and provides a user-controlled backup system.

If you realize that all of your source code editing for the last three weeks was based on some disastrous misconception, you can quietly back up to the last good version without bothering your system administrator to retrieve files from tapes, etc.

It is remarkable that this is accomplished by the SCCS utilities without the storage overhead of actually saving the full text of each version—only the differences between each version and its predecessor are saved. A side effect of the SCCS mechanism is that a source file cannot be inadvertently edited by more than one programmer.

The three principal SCCS utilities of the UNIX operating system are *admin*, *get*, and *delta*. There are a number of others, but these are the basic commands that account for 99% of SCCS usage. *admin* is used to create a basic SCCS source file from an ordinary source file. For example,

```
admin -imyfile.c  s.foofram.c
```

takes the source from the file myfile.c and creates an SCCS file named s.foofram.c. (All such SCCS files must begin with "s."). This file is an ASCII file which contains the text of myfile.c along with information about when it was created, by whom, etc.

To retrieve the text stored in s.foofram.c, it is necessary to use the *get* command. The command

```
get s.foofram.c
```

creates a file in the current directory called foofram.c containing the exact text that was *admin*'ed. If we want to update the SCCS file, the "-e" option is used with *get*. The command

```
get -e s.foofram.c
```

also creates a file in the current directory called foofram.c. The difference is that now foofram.c may be modified and returned to the file s.foofram.c.

The *delta* command is used to return an updated version of a file to its SCCS file. The command

```
delta s.foofram.c
```

removes the file foofram.c from the current directory and replaces it into the file s.foofram.c. The *delta* command also prompts the user for commentary to be stored in the SCCS file along with the updated text.

To see how these commands work together, consider the following sequence:

```
admin -imyfile.c  s.foofram.c
get -e s.foofram.c
...
...
< edit foofram.c >
...
...
delta s.foofram.c
```

The file s.foofram.c now contains both versions of the source file—the original and the edited version. To retrieve the original, type

```
get -r1.1 s.foofram.c
```

To retrieve the modified version, type

```
get -r1.2 s.foofram.c
```

The *get*/*delta* process may be repeated as many times as desired, storing an entire chain of versions of the original file in the single file s.foofram.c The number after the "r" in the *get* command is of the form release.level. The level goes up by one for each *delta*, while the release may be incremented whenever desired, for example, after a major software release. Many other variations are possible, such as retrieving a file version based upon date. It is also possible to create a branched, tree-like structure within the SCCS file, but we won't go into that here.

An additional control feature provided by SCCS is file locking. One of the known deficiencies of the UNIX operating system is that there is no built-in mechanism for coordinating several users who want to write to the same file.

Typically, if two users edit the same file simultaneously, neither will be notified of the other and the first to write to the file will lose his or her modifications when the second writes over the work. SCCS prevents this by creating a lockfile called a p-file in the directory containing the SCCS file whenever a *get* with the -e option is issued. As long as this p-file is present, the source file may not be checked out again for editing. When the source file is *delta*'ed back in, the p-file disappears, enabling further *get*s for editing.

The most interesting feature of the operation of the SCCS system is that a sequence of revisions of a source file may be saved in an SCCS file that is much smaller than the sum of the sizes of all files in the sequence. Each time a new version is *delta*'ed in, the SCCS system uses the UNIX utility *diff* to generate a sequence of editing commands to transform the previous top level version into the new top level version. Only these commands are saved in the SCCS file.

When it is time to get a file from the SCCS file, these editing commands are applied to the original (1.1) version until the desired version is reached. This editing is fast since it only deletes and adds lines; no intra-line editing is done.

Finally, the SCCS system provides a control feature called "what strings". When a file such as a C language source program is put under SCCS control, it is recommended (mandatory under PSCCS) that it contain some construct such as the following:

```
static char sccsid[ ] = "%W%%Q%";
```

The strings "%W%" and "%Q%" have a special meaning to the SCCS system. When the source file is retrieved via *get*, the line is expanded as follows:

```
static char sccsid[ ] = "@(#)foofram.c
1.2 UNIX System V/68";
```

where "1.2" is the release and level number of the retrieved file, "UNIX System V/68" is a string defined by the system administrator, "foofram.c" is the name of the file, and "@(#)" is a magic string of characters whose use will be explained later.

Suppose now that the retrieved C source file is compiled and linked to create some executable file called, say, foofram. A UNIX utility called *what*, when applied to any file, searches the file for all occurrences of the magic sequence of characters "@(#)" and returns whatever characters follow, up to an ACSII NULL or new-line character.

The file foofram contains, somewhere in its initialized data section, the string "@(#)foofram.c 1.2 UNIX System V/68". Thus the command

```
what foofram
```

will return

```
@(#)foofram.c 1.2 UNIX System V/68
```

In this way, executable object files can be easily examined to determine which versions of which source files they came from. (The string "@(#)" was chosen as the magic string simply because it is a string not likely to occur otherwise in a program; even if it should occur, however, no great harm is typically done.)

PSCCS enhancements

The SCCS utilities provide some assistance for an individual who needs to keep track of his or her own software development work. However, it will not meet the needs of a group of people working on the same project.

In a software project, a large number of source files must be kept in a common area. This presents (at least) two problems: first, programmers cannot remember where files are or what they are called; second, programmers must be prevented from inadvertently destroying work done by themselves or others.

To address these problems, we wrote a project control user interface to the SCCS commands. This interface provides controlled access to all the SCCS commands via new commands *pget*, *pdelta*, *padmin*, etc., analogous to the *get*, *delta*, *admin*, etc., of SCCS.

Each PSCCS command provides a controlled interface to its associated SCCS command. The PSCCS system must be initialized by the project administrator with a simple data base of SCCS file names (actually complete path names). This relieves the user of the burden of remembering the complete path names of all files in the system. If the user enters

```
pget string
```

the *pget* utility will search the data base of path names for all names that contain a string of characters as a substring. If there is only one such path name, that SCCS file is retrieved. If there is more than one matching path name, each one is presented to the user and the user must choose which one he or she wants.

The system actually contains some additional refinements. The user may speed up the search by specifying in the environment or on the command line one of several general groups of utilities the user is interested in. This solves the problem of having the user remember the path name of all files under the system—the user need only remember some substring of that path name. If the user wants to retrieve a version of the SCCS file /port/port/src/cmd/s.foofram.c, he or she can execute *pget foof*, for example. Neither the location of the SCCS file nor the full name needs to be specified.

The other problem—the problem of control—is easily solved by the PSCCS mechanism. In the normal SCCS environment, the ability to update the file will also give the user the ability to deliberately or inadvertently corrupt or remove the file.

Each PSCCS utility has the setuid bit in its permissions turned on. This means that whenever the program is run, the effective user ID is that of the owner of the program (the project administrator) rather than the real user. The user can now, through the PSCCS utilities, access files to which he or she otherwise has no permission. Provided the files are properly protected, the user cannot destroy the files since these capabilities are not provided through PSCCS.

The entire PSCCS interface consists of about 630 lines of C code. This includes code for the less frequently used routines *pcdc*, *pprs*, and *prmdel* as well as *pget*, *pdelta*, and *padmin*. Of this, 270 lines are for routines that expand partial path names given on the command line into complete path names via the PSCCS data base.

PSCCS and the V/68 port

This set of "project" SCCS tools was used by Motorola Inc. on their AT&T-sanctioned port of the UNIX System V operating system to the M68000 microprocessor family. In that project, over 6,000 SCCS files were used without a single incidence of corruption or loss due to programmer errors.

Unlike many other projects of similar size and complexity, almost no configuration or version control problems occurred. In the initial released product which, at the time of this writing is still the only AT&T approved microport, every object code module was created from the top "release 1" SCCS level.

Motorola's development of the UNIX System V operating system, M68000 version, continues today with successive releases for M68010 and M68020 machines all under PSCCS control. Thanks to the PSCCS tools, any release of the software can be regenerated at any time, a feat that would have seemed impossible just a few years ago.

From the programmer's point of view also, working without PSCCS would have been much more cumbersome. The primary advantage to the programmer was that it was not necessary to remember where the source files were kept or exactly what their names were in order to work with them. We are aware of no complaints or requests for enhancements to make PSCCS easier to use.

The PSCCS advantage

The SCCS utilities provide the individual programmer with a mechanism for the orderly updating of source files under the UNIX operating system. For management of the source files of a medium-to-large software project, SCCS is just not adequate.

It is possible, however, to provide an acceptable source file management facility for such a project by providing a front-end interface to the SCCS utilities. In the form presented here, the interface provides a way for the individual programmer to retrieve or update source files without having to remember their full name or location. It also provides control over which operations programmers are allowed to perform on project source files. Although much more elaborate program data bases may be constructed, we found the PSCCS system to be exactly what was needed for the management of our 6,000-file UNIX operating system port project.

Note: Next month in *COMPUTER LANGUAGE* we will present a more detailed analysis of the design criteria behind the C code used to implement PSCCS.

Luke C. Dion holds a B.S. in mathematics and in computer science from the Univ. of California at Berkeley and is part way to a M.S. in computer science from Stanford Univ. Last February Dion left Motorola, where he was project manager and responsible for Motorola's port of UNIX System V, and founded Palomino Computer Systems Inc., specializing in UNIX operating system consulting and porting.

Alan Filipski holds a Ph.D. in computer science from Michigan State Univ. He has taught at Central Michigan Univ. and Arizona State Univ. and is currently a principal staff engineer at Motorola Microsystems in Tempe, Ariz., working on the UNIX System V operating system.

Natural Language Processing and LISP

By Richard Berman

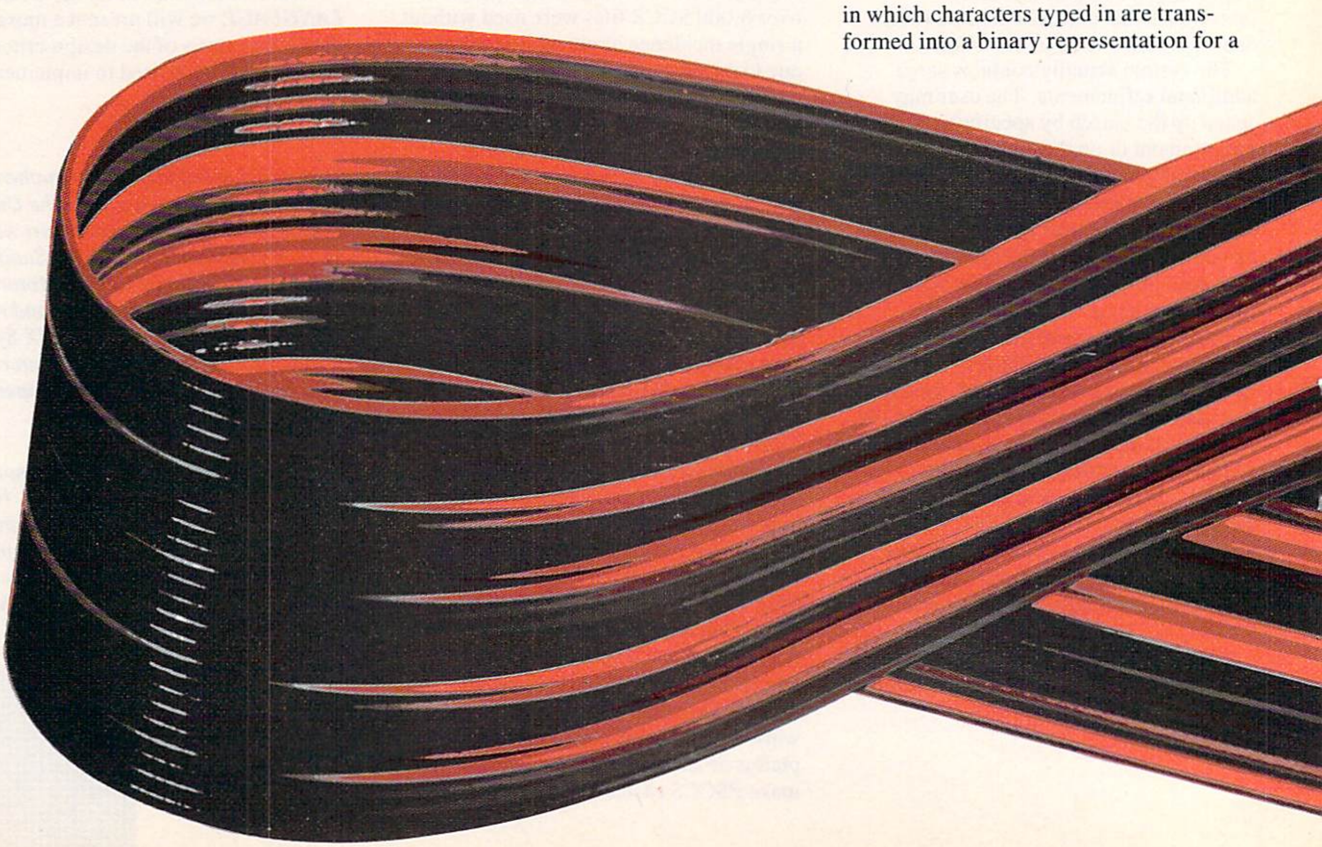
Language processing is the transformation of one set of finite symbols into another set of symbols. Right now you are processing the symbols on this page into some form that carries meaning and can be conveniently stored (or not, as you wish) in your mind. Even if you have eidetic memory, the image of this page is not the same as the meaning you assign to its content, and therefore some kind of transformation must occur.

In all cases of language processing, a transformation takes place to restructure the initial symbols into a structure that is easier to manipulate and which comes closer to representing the intent of the original statement.

Any time you must accept input in your programs some form of language processing occurs. The trivial case may be the BASIC statement

```
INPUT A
```

in which characters typed in are transformed into a binary representation for a



floating point number. On a larger scale, the statement *INPUT A* itself is an ordered set from the finite set of characters on your keyboard. Your BASIC interpreter or compiler must transform this into a form that may be executed.

At worst, this transformation may be done on the fly each time the statement is to be carried out. This is closer to the type of language processing that will be discussed in this article. The article will also cover some ideas about representation of data as well as give some advice and examples on the use of recursion to help simplify the whole process. Simple cases of natural language (e.g., English) processing will be used.

All programming examples are in LISP. These examples are short and simple, yet show the power of using lists as data structures and recursion as a processing technique. I will attempt to ease you through these pieces of code with sufficient understanding so that all will be clear.

Recursion in LISP

Recursion is a term that is receiving increased mention in technical literature. To be sure, it has always had a place in mathematics, but it is only recently that the concept has begun filtering down to all levels of computer programming.

LOGO, a language for "kids," even uses recursion in simple examples. In *Godel, Escher, Bach: an Eternal Golden Braid*[1] Douglas Hofstadter has cooked up an incredible synthesis of music, mathematics, logic, philosophy, art and artificial intelligence. One of the basic cornerstones of the book is the notion of "strange loops." This is really just a cousin to recursion, in which the only way to describe something is in terms of itself.

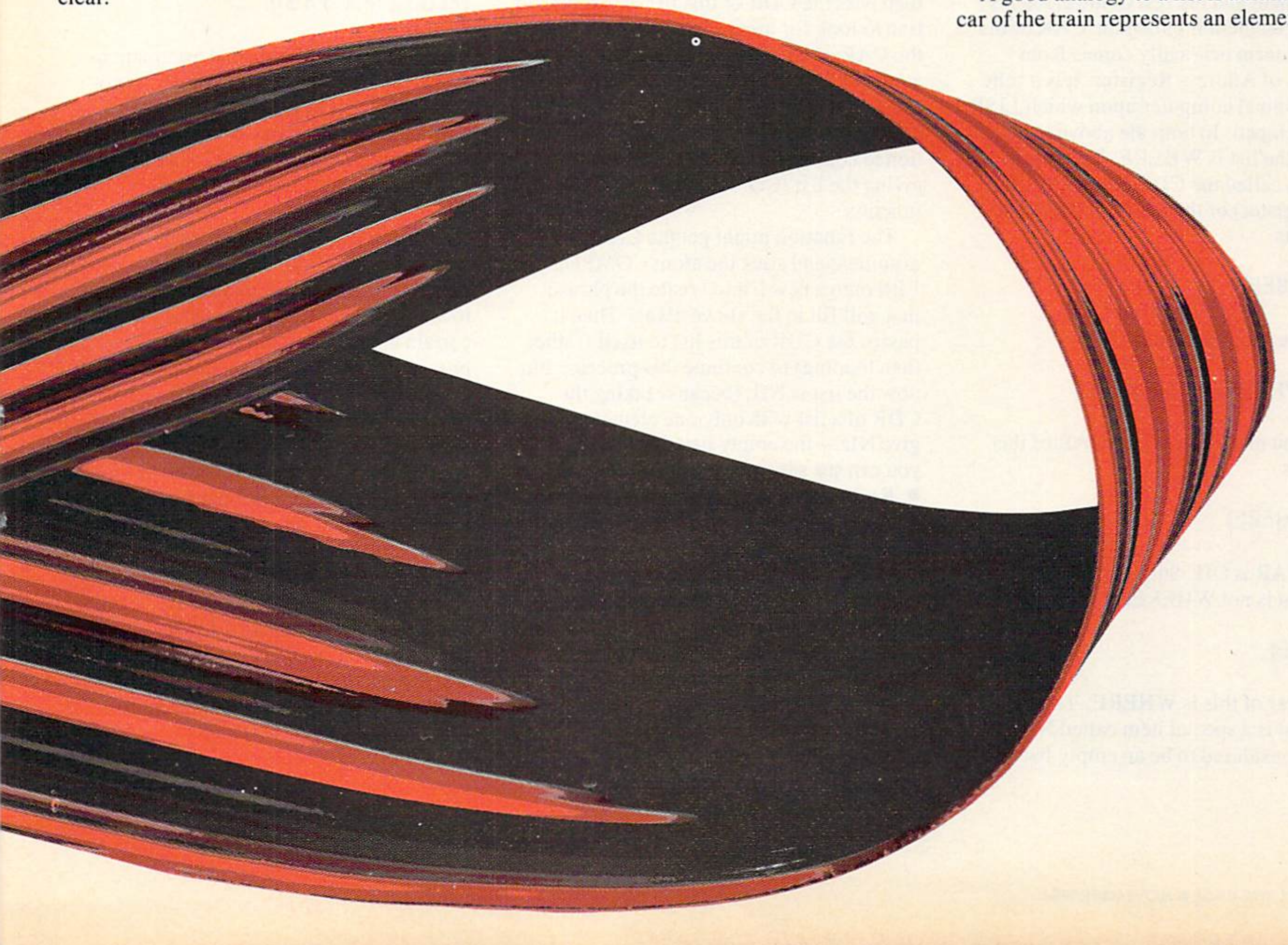
Actually, all we have in recursion is a simple function call.It is special only because the function is calling itself. On the surface, recursion may seem to be promising an eternal loop. Typically, early in a recursive function there will be a check to see if its given arguments meet some kind of terminating criteria. If so, the function returns to its caller, which often is the same function. Not all languages support recursion. Some forms of BASIC do.

LISP is not an acronym for Lots of Insipid Silly Parentheses, but it isn't hard to see why this joke originated— LISP has a lot of parentheses. LISP stands for LISt Processor.

Most programmers have some familiarity with the concept of a list. It is just an ordered grouping of items. On the surface this may sound suspiciously like an array, but there is more to a list.

Typically, anything may be an element of a list. Most languages allow only numbers or text (i.e., strings) to be the members of an array. You couldn't make an element of an array a whole other array, for example. In addition to this, a list is linked.

A good analogy to a list is a train. Each car of the train represents an element of a



list. Each one is linked up to other cars in some particular order. I could change the order by unlinking one car, or a group of cars, and relinking them elsewhere in the train. I could remove a car permanently or insert new cars at any point.

Now, if I could get an entire train into a car this would be more like a true list, because any element of a list can itself be a list. For our purposes a list might look like this

(WHERE IS THE BEEF)

and, in fact, this is what a list looks like in LISP. This list has four elements with WHERE as the first element.

The list

(WHERE (OH WHERE) IS THE BEEF)

has five elements. The second element is itself a list with two elements.

In LISP, all lists have two parts: the first element and the rest of the elements. The first element is called the CAR of the list. This term originally comes from Contents of Address Register. It is a relic of the original computer upon which LISP was developed. In both the above lists, the CAR of the list is WHERE. The rest of the list is called the CDR (Contents of Data Register) of the list. The CDR of the first list is

(IS THE BEEF)

and of the second list it is

((OH WHERE) IS THE BEEF)

and as you might guess, the CAR of this last list is

(OH WHERE)

and its CAR is OH. Surprisingly, the CDR of this list is not WHERE. It is the list

(WHERE)

and the car of this is WHERE. The CDR of this list is a special item called NIL. NIL is considered to be an empty list. It has an important purpose, as it indicates the end of a string of CDRs. Note that the CAR of this list (in this instance) is not a list—no parenthesis. In LISP parlance, anything that is not a list is called an atom because it is indivisible, at least by CAR and CDR.

As an example of the use of NIL, suppose you were writing a simple program to simulate a psychologist interrogating the user[2]. The user has entered

(I HATE COMPUTERS)

and for the purposes of our program, we wish to massage this text based on recognition of the key phrase I HATE. Perhaps we want to ask a question like "What bothers you about _____" where the blank is filled in by the object of the user's sentence, in this case (COMPUTERS).

In scanning this sentence, you might first take the CAR of this list and find the atom I. Recognizing it for a keyword, you then pass the CDR of this list onto a function to look for more. This function takes the CAR of its given value and sees the atom HATE. Another keyword! It puts it all together for the key phrase (I HATE) and passes the CDR of its list onto a function to continue parsing. This means giving the list (COMPUTERS) to this function.

The function might get the CAR of its argument and stick the atom COMPUTERS onto a new list to create the phrase that will fill in the above blank. Then it passes the CDR of this list to itself (rather than looping) to continue this process. But now the list is NIL (because taking the CDR of a list with only one element will give NIL—the empty list). At this point you can see several aspects of recursion:

- Recursion can be used in place of loops
- There must be a condition to terminate the recursive process.

In this case, the function checks for NIL and returns rather than appending NIL onto the list it is creating.

Listing 1 shows a sample set of functions to carry out a simple pattern-directed parsing of a sentence like the previous one. It is composed of three functions. Before going into them, we need to know more about LISP so that we can understand these definitions more clearly.

First off, you can clearly see by looking at the first definition that there is some kind of structure here. Certain lists line up with other lists. In LISP, indentation is used as it is with C or Pascal to indicate the structure of an expression. Also, you can see that a function in LISP is made up of a list, just as the sentences given as examples were.

LISP works in polish notation. For example, to add two numbers you would enter (PLUS 1 2) which would return the result 3. The CAR of this list is the atom PLUS. The CAR of each successive CDR is an argument. That is, the CDR of the expression is the list (2 3) and the CAR of this is 2. The next CDR is (3) and the CAR of this is 3. So 2 and 3 are arguments to *PLUS*.

Just as there was a list as an element of the sentence above, an argument of a function may be a list rather than an atom. For example,

(PLUS 2 (PLUS 4 5))

would yield 11. The second argument to *PLUS* is a list. The LISP evaluation process calls itself to get the result of this argument so that it may be added to 2. LISP is inherently very recursive in the execution of lists.

Sentence processor example

Let's go back to our simple sentence processor in Listing 1. Remember the goal is to take a sentence and, by recognition of certain keywords or phrases, extract other phrases that we can use to construct a new output sentence and so simulate a patient/psychiatrist dialogue.

By examining the first function, we can see that it is a list whose CAR is the atom DE. *DE* is a function (remember, if a list is to be executed, the CAR is a function) used to define new functions. The first argument is the name of the function being defined. In this case we are defining a function named *PARSE-FORM*. The second argument to *DE* is a list of atoms that represents the arguments of the new function. This is similar in concept to the X in the BASIC expression DEF FNA(X) = . . .

The arguments for *PARSE-FORM* are FORM and SENTENCE. The SENTENCE will be a list, like (I HATE COMPUTERS). The FORM will be used to control the flow of the parse routines. For the above example, the FORM might be (I HATE *). This FORM might be one of many in our program that will be used to attempt to decompose the sentence. These functions will try to match all the words in the SENTENCE with the corresponding words (atoms) of the FORM.

The functions shown will detect the * atom in the FORM and use it to mean "match any number of words and save this phrase in a special place." In particular we want the word COMPUTERS to get extracted from the sample input sentence so that we may form an output sentence like WHY DO YOU HATE COMPUTERS.

The remaining arguments given to *DE* define the body of the new function. Let us examine this more closely. From the indentation, it is evident that the body of *PARSE-FORM* is a function named *COND*. This is the LISP version of *If . . . Then . . . Else*. It means "conditional." *COND* takes a list of lists of the form

```
(COND  (if-1    then-1)
       (if-2    then-2)
       (if-3    then-3)

       (if-n    then-n) )
```

In sequence, if-1 is evaluated. If it returns any non-NIL value, then-1 is evaluated and its result is returned as the result of the *COND*. If if-1 resulted in NIL, then we skip then-1 and move onto the if-2, then-2 pair. This goes on until either a non-NIL "if" occurs, or until we run out of expressions. *COND* will return NIL if no "if" is satisfied.

If *PARSE-FORM* returns NIL, it will mean that the given FORM did not match the SENTENCE. In LISP, NIL is used to indicate the false condition.

An atom named T is often used to mean true, but in general any non-NIL value is true. Now, back to *PARSE-FORM*.

PARSE-FORM will take the CAR of FORM and check if it is the atom *. If it is, it calls another of our functions called **PARSE*. This will match all words up to the next atom in FORM. For example, the form

(I * YOU)

will match any of the sentences

(I LOVE YOU)
(I MUST TALK WITH YOU)
(I CAN'T RECALL WHAT THEY SAID
    ABOUT YOU)

```
(DE PARSE-FORM (FORM SENTENCE)                  ; main parsing function.
    (COND ((NULL FORM) (NULL SENTENCE))         ; termination check.
          ((EQ( CAR FORM)( QUOTE *))            ; phrase matcher?
           (*PARSE( CDR FORM) SENTENCE))        ; yes - parse phrase.
          ((EQ( CAR FORM)(CAR SENTENCE))        ; equal words?
           (PARSE-FORM( CDR FORM)( CDR SENTENCE))))) ; yes - parse rest.


(DE *PARSE ( FORM SENTENCE)             ; parse a phrase
     ( SETQ PHRASE NIL)                 ; initialize phrase.
     ( SETQ *END( CAR FORM))            ; *END = word that ends the phrase.
     ( *PARSE* SENTENCE))               ; parse up until *END.


(DE *PARSE* (SENTENCE)                  ; Parse until *END.
    (COND ((NULL SENTENCE)              ; More in sentence?
           ( PARSE-FORM FORM SENTENCE)) ; no - PARSE-FORM will assure FORM done
          ((EQ( CAR SENTENCE) *END)     ; end of phrase?
           ( PARSE-FORM FORM SENTENCE)) ; yes - parse rest of sentence.
          (T                            ; otherwise...
           (SETQ PHRASE
                 (APPEND PHRASE( LIST( CAR SENTENCE))))  ; add word to phrase.
           (*PARSE*( CDR SENTENCE)))))  ; parse rest of phrase and sentence.
```

Listing 1.

ACTIVE TRACE

"Software that lives up to its promises. *When a Basic program doesn't work the way you want it to, this package . . . will help you track the problem down . . . Scope is a tool for the beginning, advanced, or professional programmer, and it begins where the cross reference maps leave off."*

Howard Glosser, *Softalk for the IBM Personal Computer* July '84, pp 120-121

***"Extremely useful program . . .** Anyone doing much programming in Basic should appreciate Active Trace a lot."*

Jerry Pournelle, *Byte Magazine* April '83, p 234

***"A marvelous Basic programming aid . . .** It's just amazing to watch a program you wrote run under Scope, and debugging becomes if not trivial, then at least doable"*

Thomas Bonoma, *Microcomputing*, Dec. '83, p 22

*"... **a really neat utility . . .** designed to untangle even the most convoluted Basic program. . . . The documentation is almost worth the price of the package."*

Susan Glinert-Cole, *Creative Computing*, July '84, p 210

Active Trace will lead you through your program letting you know variable values (all variables or just those you specify) as they change. Your program's internal activity is presented on your screen, or printer, or it can be saved on disk. **It's simple, effective and works with the BASIC you already own.**

Active Trace **$79.95**

Includes Scope, XREF mapping and documentation

Active Trace is available for most **MS-DOS** and **CPM 2.2** systems and supports the special features of Brand specific versions of Microsoft Basic such as **Basica** on the **IBM-PC**.

AWARECO
Active Software

P.O. Box 695 Gualala, CA 95445
(707) 884-4019
800-358-9120(US) 800-862-4948(CA)

Active trace, Active software, and Scope are trademarks of AWARECO—CPM is a trademark of Digital Research—MS-DOS and Microsoft are trademarks of Microsoft Corporation—IBM-PC is a trademark of IBM Corp.

CIRCLE 3 ON READER SERVICE CARD

and in each case, our special cache (called PHRASE) will contain the list of words that appeared between I and YOU.

If the atom from FORM is not *, then the CAR of SENTENCE must match it. If this is not the case, we immediately return NIL. For example, the FORM (I * YOU) has a CAR of I. The SENTENCE (WHERE IS THE BEEF) has a CAR of WHERE. Since the CAR of FORM is not *, (it is I) the CAR of SENTENCE must match. But the CAR of SENTENCE is WHERE, which does not, and *PARSE-FORM* returns NIL indicating the given FORM did match the SENTENCE.

If all is well so far, *PARSE-FORM* calls itself using both the CDR of FORM and the CDR of SENTENCE to continue matching the rest of the list.

Now for the other functions used in *PARSE-FORM*. The first "if" in the *COND* statement is (NULL FORM). The *NULL* function tests to see if its argument is NIL. If so, it will return the atom T, else NIL. You could consider *NULL* to be the *NOT* function, because if the argument is true (i.e. non-NIL) it returns NIL, and if the argument is NIL it returns T.

Note that when all of FORM has been matched, passing the CDR of FORM to *PARSE-FORM* will pass NIL. So this first "if" will be true when all of FORM is matched.

The "then" clause for this is (NULL SENTENCE). This is important in catching the case where all of FORM has been matched, but there is still more to the sentence. For example, the FORM (I * YOU) without this check would match the sentence (I LOVE YOU BUT I MUST LEAVE YOU), leaving everything after the first YOU in the sentence unchecked. So when FORM has all been matched, *PARSE-FORM* will return T if SENTENCE is also NIL (i.e., has all been matched). If there is anything left in SENTENCE, *PARSE-FORM* will return NIL meaning the FORM was unacceptable. This shows also that *PARSE-FORM* will return T if the FORM and the SENTENCE are exact matches.

If there is still more in FORM, the (NULL FORM) expression returns NIL, and *COND* will skip to the next conditional. This is (EQ(CAR FORM) (QUOTE *)). *EQ* is a function that compares two items. If they are identical, *EQ* will return T, otherwise NIL. This "if" is comparing the first element of FORM with the atom *, the special phrase matching atom. The function *QUOTE* used here prevents * from evaluating. That is, we don't want to compare (CAR FORM) with the value of the variable named *, we want to compare (CAR FORM) with the literal atom itself. *QUOTE* prevents the evaluation process.

If the special match is indeed found, **PARSE* is called to match up until the next atom in FORM. As a matter of fact, when **PARSE* has matched up to that point, it doesn't return to *PARSE-FORM*. Instead it calls *PARSE-FORM* to continue matching the remainder of FORM and SENTENCE. If (CAR FORM) is not *, the last "if" is tried. This simply compares the first element in both FORM and SENTENCE. If they are not the same, there are no more "if" clauses to try and *PARSE-FORM* will return NIL. If the match succeeds, then *PARSE-FORM* is called to match the remainder.

As you can see, these functions form a tightly knit recursive system. Each is important to the other. They call each other and tend to call back to an earlier function rather than returning back, until their terminating condition is met. It may be illuminating to run through the process manually on a piece of paper. Any complications should resolve—just be careful to keep track of the current value of the arguments to each function as they are local to that function. Recursion actually simplifies the parsing process.

These short and simple functions replace a lot of looping and flag setting/checking. Once the concept is clear in your mind, functions like these become more obvious as the solutions to certain types of problems. Rather than describe in detail what **PARSE* and **PARSE** are doing, I will give you the necessary vocabulary to understand each and also provide some hints. Consider it an exercise to fully understand them. Here is the description of the remaining functions used in these functions.

SETQ is like the *LET* statement in BASIC. It takes two arguments. The first must be an atom. In LISP atoms may be used as variables. For example, in the function *PARSE-FORM* the two arguments FORM and SENTENCE, local variables, are atoms. The second argument to *SETQ* is an expression which is evaluated. The result becomes the value of the atom. In the expression

```
(SETQ X 1)
```

the first argument X is the atom whose value is to be assigned. The second argument is evaluated and is 1. So the value of X will be 1. Any expression may appear as the second argument to *SETQ*. For example

```
(SETQ Y (PLUS X 2))
```

would assign 3 to Y.

APPEND takes any number of arguments and makes a single list from them by concatenation. For example

```
(SETQ X (QUOTE(A B C D)))
(SETQ Y (QUOTE(1 2 3)))
(SETQ Z (APPEND X Y))
```

would cause Z to be given the value (A B C D 1 2 3). This is similar to the concatenation of strings.

LIST makes a list from its arguments also but does not concatenate. Rather, each argument becomes an element of a list. Using X and Y from above,

```
(SETQ Z (LIST X Y))
```

would set Z to the list ((A B C D) (1 2 3)). Note the difference. We do not have one long list but a list of *n* elements, where *n* is the number of arguments given to LIST.

If *LIST* is given only one argument, it still makes a list from it. (LIST 20) would result in (20)—a list with one element.

That is all for the vocabulary. With this and the following hints (and some persistence) you can understand the sample functions.

Hints

Note that the atoms PHRASE and *END in *PARSE and *PARSE* are global by virtue of the fact they are not local arguments. This means any function that changes the value of these variables makes those values available to any other function. No declaration of globalness is needed.

We are using PHRASE as our cache for the list of words matched by *. The function *PARSE is used to initialize the cache and calls *PARSE* to do the dirty work. When *PARSE-FORM* returns, the value of PHRASE will be the phrase matched (if the form was a successful match of the sentence).

Don't get confused because the functions use the same name for a local variable. The variable contains a unique value for the function call. When *PARSE-FORM* calls *PARSE, even though the argument names are the same they are in different areas. When *PARSE returns to *PARSE-FORM*, no matter what *PARSE did to FORM and SENTENCE they will be the same in *PARSE-FORM* as when *PARSE was called.

An important point: you may notice that the definition of *PARSE has the name, list of arguments, and body just like *PARSE*, but that the body is not one list but three complete lists. In this case, each list is executed in sequence, and the result of the last list is the result of the function. This is useful for initializing variables, as it is used here. Likewise after the T "if" in *PARSE* there is a multiple list for the "then" portion. Each is executed in order and it is the value of the last expression that is returned.

This feature is called implicit *PROGN*, because *PROGN* is a LISP function that takes any number of expressions and evaluates them, returning the result of the final expression.

Expert systems

There is much to know about lists, recursion, language processing and LISP. This introduction only hints at the tip of the proverbial iceberg. The modern emphasis in language processing is to use the technologies embodied in the field of expert systems[3].

In this approach, language processing is broken into stages, such as typed-in text to text broken up into annotated root

"This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
— Dr. Dobb's Journal

POWER PACKS™

COMPLETE SOURCES

☐ **PACK 1: Building Blocks I** $149
250 Functions: DOS, Printer, Video, Asynch

☐ **PACK 2: Database** $399
100 Functions: B-Trees, Variable Records

☐ **PACK 3: Communications** $149
135 Functions: Smartmodem™, Xon/Xoff, Modem-7, X-Modem

☐ **PACK 4: Building Blocks II** $149
100 Functions: Dates, Text Windows, Pull-down Menus Data Compression

☐ **PACK 5: Mathematics I** $99
35 Functions: Log, Trig, Square Root

☐ **PACK 6: Utilities I** $99
Archive, Diff, Replace, Scan, Wipe (Executable Files only)

Lattice™, Microsoft™, DeSmet™, CI-86™ Compilers on IBM PC/XT/AT™
Small and Large Memory Models.
Credit cards accepted
($7.00 handling/Mass. add 5%)

SOFTWARE HORIZONS INC.

165 Bedford Street
Burlington, Mass. 01803
(617) 273-4711

NOVUM ORGANUM

CIRCLE 25 ON READER SERVICE CARD

words. From there, the sentence is analyzed syntactically. The next stage might be to use dictionary information to represent the meanings in the sentence. Then another stage would be to carry out the request or check the validity of the information.

Each stage is an expert system in itself, utilizing a large amount of specialized information about its own task. It will transform its input set into the input set for the next stage. In some systems, there is no clear division between stages. Information from each is used to help the other. It may be necessary to understand some of the syntax of the sentence in order to determine the root words. Before fully deciding on the syntax, some understanding of the meaning being conveyed is needed. So there are interdependencies.

The use of lists is vital to all current artificial intelligence research. They enable dynamic manipulation of information and are often used to simulate more structured information.

For example, you might make a "frame" where each slot was filled by an element of a list. If the element was an atom, that is the value for that slot (like a field in a data base record). But if the item is a list, it is evaluated to determine the contents of the slot. So the slot has a runtime value that may (and probably does) depend upon other slots. This is like a symbolic spread sheet.

As much thought is defined in subterms of itself (e.g., you consider a goal and break it up into sub-goals, etc., until do-able actions are discovered), recursion is used as the most natural way to much processing of information in intelligent systems. And throughout all this, LISP has remained the primary language for experimentation. It has many simple and powerful features that bring symbol manipulation into the realm of the computable. ■

1. Hofstader, Douglas R. *Godel, Escher, Bach: an Eternal Golden Braid.* Basic Books, N.Y., 1979.
2. Weizenbaum, Joseph. "ELIZA." *CACM* 9. (1966) pp.36-45.
3. Hayes-Roth, Frederick, Waterman, Donald A., Lenat, Douglas B., editors. *Building Expert Systems.* Addison-Wesley, Reading, Mass., 1983.

Richard Berman attended California State Univ. at Northridge, Calif., from 1974 to 1977, and has been doing artificial intelligence research since then. He is currently working for Rising Star Industries.

TOTAL CONTROL:

FORTH: FOR Z-80®, 8086, 68000, and IBM® PC

Complies with the New 83-Standard

GRAPHICS • GAMES • COMMUNICATIONS • ROBOTICS
DATA ACQUISITION • PROCESS CONTROL

- ***FORTH*** programs are instantly portable across the four most popular microprocessors.
- ***FORTH*** is interactive and conversational, but 20 times faster than BASIC.
- ***FORTH*** programs are highly structured, modular, easy to maintain.
- ***FORTH*** affords direct control over all interrupts, memory locations, and i/o ports.
- ***FORTH*** allows full access to DOS files and functions.
- ***FORTH*** application programs can be compiled into turnkey COM files and distributed with no license fee.
- ***FORTH*** Cross Compilers are available for ROM'ed or disk based applications on most microprocessors.

Trademarks: IBM, International Business Machines Corp.; CP/M, Digital Research Inc.; PC/Forth + and PC/GEN, Laboratory Microsystems, Inc.

FORTH Application Development Systems include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities and 200 page manual. Standard random access files used for screen storage, extensions provided for access to all operating system functions.

Z-80 FORTH for CP/M® 2.2 or MP/M II, $100.00; **8080 FORTH** for CP/M 2.2 or MP/M II, $100.00; **8086 FORTH** for CP/M-86 or MS-DOS, $100.00; **PC/FORTH** for PC-DOS, CP/M-86, or CCPM, $100.00; **68000 FORTH** for CP/M-68K, $250.00.

FORTH + Systems are 32 bit implementations that allow creation of programs as large as 1 megabyte. The entire memory address space of the 68000 or 8086/88 is supported directly.

PC FORTH +	$250.00
8086 FORTH + for CP/M-86 or MS-DOS	$250.00
68000 FORTH + for CP/M-68K	$400.00

Extension Packages available include: software floating point, cross compilers, INTEL 8087 support, AMD 9511 support, advanced color graphics, custom character sets, symbolic debugger, telecommunications, cross reference utility, B-tree file manager. *Write for brochure.*

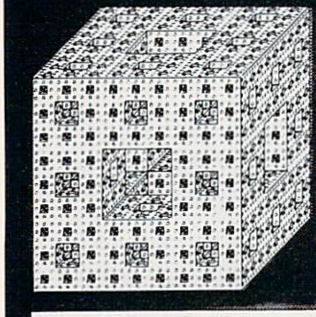
VISA

LMI

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to ***(213) 306-7412***

MasterCard

CIRCLE 35 ON READER SERVICE CARD



WALTZ LISP(TM)

The one and only ***adult*** *Lisp system for CP/M users.*

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with *Franz* (the Lisp running under *Unix*), and is similar to *MacLisp.* Waltz is perfect for Artificial Intelligence programming. It is also most suitable for general applications.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix • True dynamic character strings. Full string operations including fast matching/extraction. • Flexibly implemented random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), nlambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Virtual function definitions. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Assembly language interface. • Over 250 functions in total. • The best documentation ever produced for a micro Lisp (300+ full size pages, hundreds of illustrative examples).

Waltz Lisp requires CP/M 2.2, Z80 and 48K RAM (more recommended). All common 5" and 8" disk formats available.

ProCode(TM)
INTERNATIONAL
15930 SW Colony Pl.
Portland, OR 97224

Unix* Bell Laboratories.
CP/M* Digital Research Corp.

Version 4.4
(Now includes Tiny Prolog written in Waltz Lisp.)

$169*

*Manual only: $30 (refundable with order). All foreign orders: add $5 for surface mail, $20 for airmail. COD add $3. Apple CP/M and hard sector formats add $15.

Call free **1-800-LIP-4000** Dept. #13
In Oregon and outside USA call 1-503-684-3000

CIRCLE 53 ON READER SERVICE CARD

Building Portable Programs

By Mark Grand

When building a piece of software, frequently one of the design goals is portability: the ability to run a program with different hardware and/or a different operating system than the program was originally developed for. When porting software, differences not originally anticipated will often need to be smoothed over because there is so much variation in the way CPUs and peripherals work.

The purpose of this article is to point out some of the pitfalls in writing portably and ways to avoid them.

This article is written in terms of the C language, yet most of the issues discussed are also applicable to programs written in other languages.

One of the strengths of C is that it comes with a pre-processor that provides macros and conditional compilation. If you are using a language that does not have a pre-processor with these features, after reading this article you may want to buy or build one.

The first impediment you may encounter is a lack of standardization in the language you are using. There is no standard BASIC. Even C, which is touted as promoting portability, is not totally standard. Some implementations omit features (float, separate compilation, and bit fields, for example) specified by Brian Kernighan and Denis Ritchie in *The C Programming Language.* Features have been added to the language since this book was published. In general, try to stick to a subset of the language you are working with that is standard and universally available.

A strategy that seems to work well in dealing with this sort of problem is one of abstraction. An application of this is illustrated in Listing 1. By replacing the addition of all *float* numbers $(x+y)$ with the construct *fadd(x, y)*, our problem becomes readily solved. If *float* is implemented, then *fadd* is defined as a macro that expands to the standard addition construct. If *float* is not available, it can be defined by a suitable *typedef*, and *fadd* becomes a subroutine call. We might also want to make *fadd* a subroutine even if *float* is implemented so we can make use of a co-processor not supported by the compiler.

The biggest advantage to this abstraction strategy is that we can define all our environment-dependent abstractions in header files. The body of our code can be the same wherever we move it. The compiler, machine, and operating system dependencies can be confined to header files plus whatever procedures need to be written to support an abstraction in a particular environment.

Keeping the compiler, machine, and operating system dependencies in separate files further simplifies things. If we have x different CPUs running y different operating systems then we need $x+y$ separate header files as opposed to $x*y$ combined header files.

It will generally be desirable to process the header files in the following sequence:

- Machine dependencies
- Operating system dependencies
- Compiler dependencies.

This sequence is recommended because some operating system characteristics may depend on the hardware, and some compiler characteristics may depend on both the hardware and the operating system.

```
Replacing x+y with fadd(x, y) adds generality

#if FLOAT_IMPLEMENTED
#define fadd(a, b)      (a)+(b)
#else
extern float fadd();
#endif
float x,y,z;
x = fadd(x, y);
```

Listing 1.

The beginning of a machine-dependent header file is shown in Listing 2. It contains:

- The type and model of the CPU
- The direction in which stacks grow. (Some machines have *PUSH* instructions that increment a stack pointer; others have *PUSH* instructions that decrement a stack pointer. This information can be useful when writing procedures that can take a variable number of parameters)
- The number of bits normally used to store characters
- On some machines, if strings are packed into integers the characters are packed so that the character furthest to the left occupies the most significant bits; on other machines this character occupies the least significant bits.

The beginning of an operating system-dependent header file is shown in Listing 3. It contains the:

- Name of the operating system
- Version of the operating system
- Character set in common use. Note that ASCII and EBCDIC are not the only possibilities
- Radix that is generally used to print addresses. It is annoying to have your own debugging code printing addresses in hex when all the tools that come with the operating system speak only octal.

Many operating systems organize main storage into pages. Do not assume that the page size will be a constant expression. Some machines, such as the Burroughs 6000 and 7000 series, divide main storage into pages of nonuniform size.

The sequence of characters (if any) customarily used to indicate the end of a line of text varies with the operating system. Typical end-of-line indicators are <line-feed> (aka <new-line>), <carriage-return><line-feed>, <carriage-return> and end-of-record.

```
Beginning of a machine dependency file

#define PROCESSOR_NAME                    "VAX"
#define PROCESSOR_TYPE                    VAX
#define PROCESSOR_MODEL                   780
#define STACK_GROWS_DOWN                  TRUE
#define BITS_PER_CHAR                     8
#define CHARS_PACKED_LOW_TO_HIGH          TRUE
#define FLOAT_IMPLEMENTED                 TRUE
```

Listing 2.

```
Beginning of an operating system dependency header file

#define OS_NAME                         "VMS"
#define OS_NUMBER                       VMS
#define OS_MAJOR_VERSION                3
#define OS_MINOR_VERSION                6
#define CHARACTER_SET                   ASCII
#define USUAL_RADIX_FOR_ADDRESSES       HEX     /* for printing addresses */

#ifndef PAGE_SIZE                               /* may already have been
                                                   defined in machine
                                                   dependency file */
#define PAGE_SIZE                       512
#endif

#define FILE_NAMES_ARE_CASE_SENSITIVE   FALSE
#define HAS_FILE_VERSIONS               TRUE

#define EOL                             "\n"    /* end of line */

#define is_null(c)                      (c == '\0')
```

Listing 3.

The implemented language

Most language standards are very specific about how explicit control constructs such as *IF*, *WHILE* and *GOTO* are supposed to work. These can generally be relied upon to be the same for most language implementations. Implicit control structures are less reliable.

The Pascal standard does not specify whether or not both operands of the *and* and *or* operators will always be evaluated. If you are working with an implementation of Pascal that does not evaluate the right operand of *and* unless the left operand is true then the following will work quite well:

```
if (pt < > nil) and (succ(pt↑ .ss) < > n)
    then ...
```

For implementations of Pascal that always evaluate both operands of *and*, this construct is likely to produce some strange run-time errors.

Most languages do not guarantee the order in which the actual parameters to procedure call are evaluated. After foo($x = 2, x = 3$), it is not defined whether x will be 2 or 3. Some languages do not define whether the right or the left operand of a given operator is to be evaluated first.

If you are working in such a language, do not assume that you will be able to determine these details for a given compiler by seeing what it does with simple expressions. Optimizing compilers may decide on an order of evaluation based on such arcane things as a particular value being available in a register. The decisions that optimizing compilers make on these matters may even vary with the version of the compiler.

Arithmetic can be another problem area. The range of integers varies with the type of CPU. Some languages guarantee a minimum range for integers. Other languages provide a built-in function or constant that returns the maximum value representable.

If not provided by the language we can provide this information in a machine-dependent header file (see Listing 4). Even the representation of a binary integer is not totally standard. Some manufacturers, such as UNIVAC, are building computers that use ones complement, rather than the more common twos complement notation.

Floating point is even worse. It varies in both range and precision. Also the radix of the faction is not necessarily two. IBM's 370 architecture provides an interesting variation. The base of the exponent is 16. The exponent can range from −65 to 63. The radix of the fraction is 16.

A file by any other name . . .

At least 98% of all the programs I have ever written make explicit use of some feature of the operating system it was written within. The operating system feature that seems to be most universally used explicitly is file management (aka BDOS, file system, etc.).

This feature has many names and even more differences in the way it is implemented. File I/O is part of the definition of many languages. This provides some small amount of consistency from one operating system to the next.

No I/O facilities are included in the definition of the C language. There is a set of procedures that is usually provided to perform some of the higher level I/O functions. When C is implemented outside of a UNIX environment, usually additional routines try to provide lower level file services in a manner compatible with UNIX. Depending on the particular operating system, full compatibility may not be possible.

The first operation we usually perform on a file is to identify it to the operating system. To identify a file, we generally need to provide one or more of the following pieces of information.

1. On what device is the file located? For some devices such as a terminal, this may be the only information required.

2. For disks, there is usually some specification to indicate a subset of the files. In the terminology of many operating systems, this is called a directory. CP/M calls this a user number.

3. A file ID. On most operating systems there is a provision for following a primary file ID by one or more supplemental IDs. The operating system terminology may refer to these as qualifiers, types or extensions. If the device is a tape then the file ID may consist only of a number indicating the file's relative position on the tape.

4. A version or generation number. When a new file is created that has the same name as an existing file, most operating systems cause one of two things to happen: the old file disappears and is replaced by the new file or, if the operating system (such as TOPS-20 or VMS) supports version numbers, the old file remains and the new file is assigned a higher version number than the old file.

Most operating systems supply defaults for items 1, 2, and 4. Most modern operating systems allow all the listed items to be specified as part of a file name. Some operating systems insist on items 1, 2, or 4 being specified separately from the file name. Some operating systems consider

```
Arithmetic machine dependencies

#define TWOS_COMPLEMENT                       TRUE
#define MAX_SHORT,MAX_INT                     32767
#define MAX_LONG                              2147483647

#define FLOAT_MAX_EXPONENT                    127
#define FLOAT_MIN_EXPONENT                    -127
#define FLOAT_EXPONENT_BASE                   2
#define FLOAT_FRACTION_RADIX                  2
#define FLOAT_FRACTION_PRECISION              23
#define FLOAT_DECIMAL_PRECISION               6          /* really 6.8 */
```

Listing 4.

"myfile" and "MyFile" to be the same; others do not. While most operating systems place length restrictions on the primary and supplementary file IDs, it seems to be a safe assumption that a given operating system will allow primary file IDs up to six alphanumeric characters in length.

What is one to do about this diversity in file name formats? One solution is to ignore the problem entirely and treat file names as arbitrary strings. This may be workable if all file names are either provided by the user or are constant and can be defined as macros in an operating system-dependent file. A drawback to this approach is that in many environments users expect programs to supply defaults for parts of file names: if I ask a FORTRAN compiler to compile a file named "crunch", I may expect it to know that I really meant "crunch.for".

If ignoring the structure of file names does not seem feasible, we need to be aware of the components of file names. This can be solved by adding two procedures to the operating system dependent header file. The purpose of the first procedure would be to parse a file name and fill a structure with the pieces of information extracted from the file name.

It may be desirable to have this procedure issue error messages about improperly constructed file names. This is useful when incomprehensible messages would otherwise be produced by the operating system. It also provides an opportunity to comment on bad file names without having to try to open a file.

A second procedure is needed to construct file names. This procedure would be passed a structure from the sort produced by the file name parser and return a string containing a file name.

Given these file name structures, it becomes a much simpler matter to determine if part of a file name has been omitted and to supply a default value for part of a file name.

What can we do with a file?

The three file organizations most commonly supported are sequential, random, and keyed.

Sequential files are the most universally supported and are generally available on all devices. The only thing to beware of when dealing with sequential files is that an end-of-file error may be encountered while writing a file. Some operating systems have the notion that the size of a file is known before the file is created and will allocate a fixed amount of disk space on file creation. Other operating systems allow you to create a file without knowing how large it is going to be, but after the file is closed the first time its size becomes fixed, and you can't append to the end of it. This is usually the case with files on magnetic tape. These file size problems apply to all file organizations.

Random access files are supported almost as universally as sequential. Unfortunately, when you tell an operating system that you want to read a random file starting at an offset 300 from the beginning of the file, you will get different results depending on the operating system and possibly how the file was created.

Operating systems most commonly consider files to consist of either a stream of bytes, words, or multi-byte records. Some operating systems support all three. Some support only one of these. If a file is considered to be a stream of records, then the records may be fixed length (all the same) or variable length (all different).

If a file has fixed length records, it is possible to calculate which record the *n*th byte or word is located in. If a file has variable length records, the only way to find the *n*th byte or word is to read the file from its beginning, counting until you get where you want to be. It is always possible to access files created as random files in a sequential manner. Not all operating systems will allow files created as sequential to be processed as random.

A number of operating systems allow holes in random files. This means that if you create a random file and write 1K of information at the beginning of the file and another 1K starting at an offset of 300K from the beginning of the file, the total disk space consumed by the file is 2K rather than 301K. Unfortunately, many operating systems do not support this feature.

The keyed file organization is not as universally available as the sequential or random. The keyed file organization has many synonyms (ISAM and B-tree, for example). The type, number, and format available for keys varies widely. One thing to beware of is that some implementations of keyed files either do not allow sequential access or, if they do, the records may not appear in any predictable sequence.

There is an unfortunate lack of standardization in what one can do with a file on different operating systems. One way to cope is to limit use of file system features to those most universally available. Sometimes this approach can be too constraining to be practical.

Another approach is to design a file system that would be ideal for your purposes and implement it on top of existing file systems. This can be something simple, such as putting some macros or procedures in an operating system dependency file to guarantee that you will always be able to use record offsets with random files.

Your requirements may necessitate something much more complicated. The

C

Software Development

PCDOS/MSDOS

Complete C Compiler
- Full C per K&R
- Inline 8087 or Assembler Floating Point, Auto Select of 8087
- Full 1Mb Addressing for Code or Data
- Transcendental Functions
- ROMable Code
- Register Variables
- Supports Inline Assembler Code

MSDOS 1.1/2.0 Library Support
- All functions from K&R
- All DOS 2.0 Functions
- Auto Select of 1.1 or 2.0
- Program Chaining Using Exec
- Environment Available to Main

c-window™
Symbolic Debugger
- Source Code Display
- Variable Display & Alteration Using C Expressions
- Automatic Commands
- Multiple Breakpoints by Function & Line Number

8088/8086 Assembler
- FAST — Up to 4 times Faster than IBM Assembler
- Standard Intel Mnemonics
- Compatible with MSDOS Linker
- Supports Full Memory Model

8088 Software Development Package
$199.00

Includes: C Compiler/Library, c-window, and Assembler, plus Source Code for c-systems Print Utility

c-systems

P.O. Box 3253
Fullerton, CA 92634
714-637-5362

CIRCLE 16 ON READER SERVICE CARD

most important guiding principle here is to minimize the number of features that you (rather than the operating system) have to support. Remember, much of the code for this will be operating system-dependent.

Some pieces of information can be obtained about a file that have absolutely nothing to do with its internal organization. You may want to put procedures or macros in an operating system dependency file that, when passed a file name, obtain a file's:

- Full name—This may include device, directory, and version information. Most operating systems provide defaults for these. If a file name typed by a user is ambiguous and any of the file name defaults change later on during execution, the file identified by the file name that the user typed may change.
- Existence—A file's existence cannot always be determined by whether or not the file can be opened. Some operating systems provide ways of limiting access to a file to a limited set of users or allowing only the owner of a file write access but everyone read access. If you do not have permission to access the file then you will not be able to open the file. Some operating systems allow users to temporarily obtain exclusive access to a file. This mechanism may also prevent successful opening of a file even though it exists.
- Permissions—Many operating systems provide a way of specifying whether or not a particular user or class of users may have read, write, or delete access. The procedure or macro that implements this should be able to take a password as a second parameter and return an integer with bits corresponding to the permissions turned on or off.
- Availability—This should determine whether someone else has obtained exclusive access to a file. Since some operating systems differentiate exclusive read access and exclusive write access you may want the result to have separate bits to indicate whether the file is available for reading or writing.
- Creation date—Should return the time and date of the file's creation or zeros if this information is not available.
- Modification date—Should return the time and date the file was last modified or zeros if this information is not available.

Frequently other pieces of information are available about a file. The preceding ones are just some of the most universally available.

Special problems with text files

As noted before, a number of different character sequences (including end-of-record) are commonly used to indicate the end of a line. It is not uncommon to want to read a line of text and then process that line of text without having to be concerned about what terminates the line of text.

It is frequently helpful to have a procedure that reads a line of text and strips off any end-of-line characters. For the sake of discussion, let's call this procedure *line_read*. If we have defined what generally constitutes an end-of-line in an operating system dependency file, the code for *line_read* can be reused for all operating systems.

In some environments it is common practice to put text in fixed length records, one line of text to a record. This way of doing things is a legacy from an era when the keypunch was the primary means of getting text into a machine. (IBM and its competitors are still manufacturing machines that put "card images" on floppy disk and tape.) When reading this sort of file it may not be practical to treat just the end-of-record as the end of a line. It may also be necessary to consider trailing blanks a part of the end of a line, in which case *line_read* must strip off trailing blanks.

Some other generally unwanted characters sometime pop up when dealing with record-oriented text files. When text is put into fixed length records by programs (in FORTRAN, most commonly) the records are usually padded out by nulls ("\0"), though sometimes an all-ones character (ASCII "\177" EBCDIC "\377") is used.

Occasionally, superfluous control characters will turn up in text files. There are a variety of reasons for this—none of them predictable. Most often though, superfluous control characters seem to be introduced when text is moved between computers or from terminal to computer. It may be attributable to random noise, inconsistent assumptions about file formats, or even data communications protocols.

If the probability of encountering random spurious control characters is considered to be low, then it might suffice to ignore null characters used to pad out records. If files are to be read one character (as opposed to record) at a time, it will be helpful to define a macro or procedure, called *is_null*, that returns true when passed a padding character.

If the probability of encountering random spurious control characters is considered worrisome, it might be appropriate to filter all the control characters you are not specifically interested in seeing. This should be done by means of a macro in an operating system-dependent file.

While it may be possible to come up with a single expression that identifies both ASCII and EBCDIC control characters, there are some less common character codes that such an expression will most surely not work for.

DeSmet C

8086/8088 Development Package $109

FULL DEVELOPMENT PACKAGE
- Full K&R C Compiler
- Assembler, Linker & Librarian
- Full-Screen Editor
- Execution Profiler
- Complete STDIO Library (>120 Func)

Automatic DOS 1.X/2.X SUPPORT

BOTH 8087 AND SOFTWARE FLOATING POINT

OUTSTANDING PERFORMANCE
- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER $50
- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT $35
- Uses DOS .OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

Check: ☐ Dev. Pkg (109)
☐ Debugger (50)
☐ DOS Link Supt. (35)

SHIP TO: ____________________

____

____________________ ZIP ______

C WARE CORPORATION

P.O. BOX C
Sunnyvale, CA 94087
(408) 720-9696

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add $5, elsewhere add $15. Checks must be on US Bank and in US Dollars. Call 9 a.m. – 1 p.m. to CHARGE by VISA/MC/AMEX.

CIRCLE 18 ON READER SERVICE CARD

Applications Developers

"C" INTO THE FUTURE
WITH
db_VISTA

The first DBMS designed exclusively for the C language.

C is the applications development language chosen by many of the largest and most successful microcomputer software houses. Now with **db__VISTA, C** can be your development language choice, too.

db__VISTA is the database management system that helps you easily define and manage databases — no matter how complex your information structuring requirements. Features include:

- Written in C, under Unix.
- Minimal data redundancy using the network database model.
- Virtual memory disk accessing.
- Fast B*-tree indexing method for key files.
- Multiple key records — any or all data fields may be keys.
- Unlimited run-time distribution license available for $795.
- Three month extended applications support included.
- PC-Write word processor/text editor included at no charge.

AVAILABLE NOW

For Lattice C, DeSmet C, or Computer Innovations' C86 under MS-DOS, with thirty day money-back guarantee. Available soon for Unix/Fortune 32:16, Xenix/Altos 586, and CTOS/Convergent Technologies systems.

db__VISTA versions:		**Development Packages:**	
Lattice	$495	Lattice C w/db__VISTA	$795
DeSmet	495	Lattice C only	395
Computer Innovations	495	DeSmet C w/db__VISTA	595
db__VISTA Documentation	15		

RAIMA
CORPORATION

11717 Rainier Ave. South
Seattle, WA 98178
206/772-1515

CIRCLE 24 ON READER SERVICE CARD

Once you choose Lattice, our friends will C you through...

LATTICE INC.: LATTICE WINDOWS, CURSES UNIX SCREEN CONTROL LIBRARY, C-FOOD SMORGASBORD, dB-C ISAM COMPATIBLE WITH dBASE II AND III... **LIFEBOAT ASSOCIATES:** FLOAT 87 8087 SUPPORT PACKAGE, HALO GRAPHICS PACKAGE, PANEL SCREEN LIBRARY... **GREENLEAF SOFTWARE:** THE GREENLEAF C FUNCTIONS... **C SOURCE:** BASIC_C C FUNCTIONS FOR BASIC USER... **SOFTCRAFT:** BTRIEVE ISAM FILE SYSTEM, BTRIEVE ISAM NETWORK FILE SYSTEM... **BLAISE COMPUTING:** TOOLS, TOOLS2, VIEW MANAGER SCREEN PACKAGE... **MORNING STAR SYSTEMS:** PROLIBRARY, PROSCREEN... **CREATIVE SOLUTIONS:** WINDOWS FOR C... **NOVUM ORGANUM:** C POWERS PACKS, MATHEMATICS POWER PACKS, ADVANCED POWER PACKS, DATABASE POWER PACKS, TELECOMMUNICATIONS POWER PACKS W/ SOURCE... **PHACT ASSOCIATES:** PHACT ISAM LIBRARY... **RAIMA CORPORATION:** db_ VISTA DBMS... **PHOENIX:** PLINK86, PFIX86... **RELATIONAL DATABASE SYSTEMS:** C-ISAM FILE ACCESS METHOD... **MINDBANK:** V-FILE VIRTUAL MEMORY/FILE SYSTEM... **HUNTER & READY:** VRTX C INTERFACE LIBRARY... **GRAPHIC SOFTWARE SYSTEMS:** GSS DRIVERS, GSS TOOLKIT KERNEL SYSTEM... **OPT-TECH DATA PROCESSING:** OPT-TECH SORT... **ACCUDATA SOFTWARE:** C-TREE ISAM, C-SORT SORT... **TRIO SYSTEMS:** C-INDEX+ ISAM... **COMPU CRAFT:** c VIEW FORMS/WINDOW MANAGEMENT... **SCIENTIFIC ENDEAVORS:** GRAPHIC PRESENTATION SCIENTIFIC GRAPHICS... **LEMMA SYSTEMS, INC.:** C LIBRARY... **ESSENTIAL SOFTWARE, INC.:** C UTILITY LIBRARY... **SOFTWARE LABS:** C UTILITIES PACKAGE... **FAIRCOM:** C-tree BY FAIRCOM ISAM WITH SOURCE

Contact Lattice to learn how we can help your C program development.

LATTICE®

P.O. Box 3072
Glen Ellyn, IL 60138
312/858-7950
TWX 910-291-2190

CIRCLE 36 ON READER SERVICE CARD

However you choose to deal with unwanted control characters, a small wrinkle crops up when reading a file one line at a time. Between the last character of text in a file and the end of that same file may be some pad characters. If these pad characters are not processed before testing for the end-of-file, your end-of-file test may return false when you wanted it to return true. This can happen even in byte-oriented files and is least expected when it does.

I once had a program that seemed to work perfectly except for blowing up on one particular file. It turned out that this had been produced by an editor that tried to minimize the number of its write operations by packing characters into integers and then writing the integers. If the last integer written was not full, there would be trailing nulls.

Problems with character codes

Two character sets are most commonly used in the U.S.: ASCII and EBCDIC. ASCII is defined by ANSI standard X3.4. (ANSI standards can be obtained by writing to American National Standards Institute, 1430 Broadway, New York, N.Y. 10018. ANSI will provide a list of available standards and pricing information without charge.)

So far there have been three versions of this standard. The primary legacy of the original 1963 version is that some very old programs treat '~' and '|' in a manner similar to escape. Also the character code that now stands for the character '_' stood for a left pointing arrow. This is why some languages use '_' as an alternative assignment operator.

A 1965 standard was officially approved but not officially published. The standard currently in use was approved in 1968. A revision in 1977 changed some of the language to be consistent with international standards. ("7-Bit Coded Character Set for Information Processing Interchange," ISO 646-1973, also available from ANSI).

The ASCII standard provides for some variations. The standard states that the character code '\n' (referred to as line feed) "advances the active position to the *same* character position on the next line." The standard also states that, alternatively, "this character may have the meaning "New Line", a format effector that advances the active position to the *first* character position on the next line." Similar variations are permitted for vertical tab and form feed.

For those concerned about shipping their software outside the U.S., ASCII also conforms to an international standard. The international standard provides for some variation in the graphic rendition of some character codes.

The codes rendered by ASCII as '@', '[', '\', ']', '{', '|', and '}' are designated as being for "national use." That means these character codes can be, and are, used for letters in other countries' alphabets. The codes rendered by ASCII as '∧', '`', and '″' are designated as being for "supplementary national use." That means that in countries where the seven "national use" character codes are not sufficient (such as in Fed. Rep. Germany), these three characters are up for grabs.

EBCDIC is defined by IBM, its main proponent. While more than one version of EBCDIC exists, most of the differences do not affect conversion between ASCII and EBCDIC. The main problem I've encountered when converting ASCII to different versions of EBCDIC is that there are two different ways to translate the ASCII codes for '[' and ']' into EBCDIC. Because of different versions of EBCDIC, both translations are correct. If the characters either do not print or '[' prints as '<¢>' and ']' prints as '!' then you know you've guessed wrong.

Some macros for manipulating character codes you may want to define are found in Table 1.

Communicating with a terminal

Many programs concern themselves, at varying levels of detail, with terminal I/O.

Most operating systems do a competent job of managing the details involved with printing terminals and glass teletypes (non-display terminals). Usually, an application needs to know at most three (and possibly zero) things about a non-display terminal (displays can optionally be treated as glass teletypes): the terminal type, line width, and page length.

All three of these can be implemented as macros or procedures in an operating system-dependent file. If there is any possibility that you will be writing something that will communicate with more than one type of terminal, these macros or procedures should take a file as a parameter so they can know which type of terminal you are talking about.

The terminal type macro or procedure should ideally return an indication of whether the terminal is a printing terminal, a glass teletype, or a specific type of display. Often it is enough to know whether or not you are dealing with a printing terminal. Most nonprinting terminals won't overstrike characters. Most nonprinting terminals that can overstrike characters won't unless told to.

These three pieces of information are not provided by many operating systems. If a piece of information describing the environment is not available from the operating system, an alternative is to put the information in a file. Whatever format you decide upon for an environment file, it should be possible to extract a piece of information quickly.

Procedures that extract data from an environment file should have defaults that can be used if the environment file is missing. It would be a shame for an otherwise wonderful piece of software to become totally nonfunctional just because it wasn't sure of the page length.

Information from the environment

I use the term "environment" here because on some smaller systems things like the date are obtained directly from the hardware and not through the operating system. All the same, it is probably most general to put macros or procedures to implement these functions in an operating system-dependent file so that they compile conditionally. That way you can still define these things in a machine-dependent file when appropriate (see PAGE_SIZE in Listing 3).

Dates come in an incredible number of formats. One of the things you will want your date fetching procedure or macro to do is provide the same format of date no matter what environment you are in.

But what format to choose? I would suggest one of two formats (or possibly both) because they are already provided in a number of environments. One format consists of three integers: the year, month, and date. This format is most useful if your primary interest in the date is to be able to print it.

If you want to perform calculations such as determining the difference between two dates, adding a number of days to a date, determining the day of week a particular date falls on, or just comparing two dates, this second format is easier to work with. It consists of a long integer containing the number of days since Nov. 17, 1858. (This is derived from the Smithsonian Universal Astronomical Time Standard. Nov. 17, 1858, was when the Greenwich, England, observatory started maintaining the standard for Greenwich mean time.)

The time of day is most commonly represented as three (and sometimes four) numbers (hours, minutes, seconds, and possibly a fraction such as tenths or thousandths of a sec) or a single number. Single number representations of time generally are the number of some time unit since midnight. The time unit is generally no larger than 1 sec and usually smaller.

Varying time units can be dealt with in two ways. One way is to write a macro or procedure that produces the time, always using the same time unit regardless of the actual precision available. When selecting a time unit, remember that the time may not be available to greater than 1 whole sec of precision. A 32-bit-long integer can count in 1/1,000 of a sec but will overflow if you try 1/10,000 of a sec. A 32-bit unsigned integer will work for 1/10,000 of a sec.

If you need all the precision that is available on any given hardware, things become more involved. If you are not planning to execute on any hardware that uses precision greater that 1/10,000 of a sec (1/1,000 if your implementation language does not support unsigned integers), the only additional thing you need to do is to define a macro that indicates how many clock ticks there are in 1 sec. Note that this may not be a power of 10.

Some clocks are synchronized to the frequency of the a.c. power giving 60 or

to_lower	If a character is an upper case letter, it produces its lower case counterpart. The relationship between upper and lower case alphabetics varies greatly between character code schemes.
to_upper	The inverse of to_lower
is_alpha	True for character codes that correspond to letters
is_upper_case	True for character codes that correspond to upper-case letters
is_lower_case	True for character codes that correspond to lower-case letters.
next_alpha	Returns a letter of the same case as its parameter that lexicographically follows its parameter. This is needed because in many character codes the letters of the alphabet are not all assigned consecutive character codes.
prev_alpha	The inverse of next_alpha.
is_digit	Will be "true" if its parameter is a character code for a decimal digit. This is advisable because there may be some character code schemes that do not assign the decimal digits consecutive character codes in lexicographical order.

Table 1.

50 clock ticks per sec. If you are planning to run on machines that keep time to a greater precision than will fit in any data type provided by your implementation language (this is not uncommon with some of the larger minis and mainframes), things get more complicated. You will need to define a structure that is an appropriate size for containing the time and macros or procedures to compare times, add times, subtract times, etc.

Run-time libraries

The list of portability issues covered here is far from complete. I have not addressed such issues as memory management, segmented address spaces, display terminals, magnetic tape, and others because of space limitations.

Some languages come equipped with solutions to some of these issues built in. In this respect, C is lacking in comparison. The design of C necessitates that many of the features built into other languages be implemented as run-time libraries. This provides a great deal of freedom. This freedom shifts the responsibility for a number of features from the builder of the compiler to the maintainers of run-time libraries.

The burden of supporting a run-time library adequate to shielding most programs from environmental idiosyncrasies for a large number of environments is a lot of work. Doing it for your own applications is probably a duplication of effort. While I am not aware of any company that provides this sort of run-time library for a great many environments for use with the C language, there is a company in California that provides a much more than adequate run-time library for a language called MainSail.

Programs written in MainSail tend to be even more portable than programs written in C. Unfortunately, programs written in MainSail tend to be larger and slower than those written in C. To date, the smallest CPU they support is the 68000. They also support a number of minis and mainframes with a variety of operating systems.

References

1. ANSI Standard X3.4 1977. *Code for Information Interchange*.
2. ANSI Standard X3.28 1976. *Procedures for the Use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links*.
3. ANSI standard X3.64 1978. *Additional Controls for Use with ASCII*.
4. Kernighan, Brian W. and Ritchie, Denis M. *The C Programming Language*, Prentice-Hall, 1978.
5. IBM Systems. *370 Principles of Operation*, 1973.
6. Jensen, Kathleen and Wirth, Niklaus 1978. *Pascal User Manual and Report*, second edition, Springer-Verlag.
7. MacKenzie, Charles E. *C*, 1980.
8. Mulders, H. "Some Observations on the In- and Output in High Level Languages," *Sigplan Notices* vol. 18, no. 9 (September 1983): 55.
9. *VAX Technical Summary*. Digital Equipment Corp., 1980.

Mark Grand has a B.S. in computer science from Syracuse Univ., Syracuse, N.Y. He has done a lot of work with language design, text editing, and man/machine interfaces and is now working for a software house that produces a financial modeling language.

PROLOG-86 tm

Learn Fast, Experiment

1 or 2 pages of PROLOG would require 10 or 15 pages in "C."

Be familiar in **one evening.** In a **few days** enhance artificial intelligence programs included like:

- **an Expert System**
- **Natural Language** (generates dBASE display)

Intro Price: $125 for PCDOS, CPM-86.
Full Refund if not satisfied.

CONTEST: "Artificial Intelligence Concepts"

$1,000 Prize, Recognition for applications in PROLOG-86™ that teach, are clear, illustrate. Call for details. Deadline 11/31/84

SOLUTION SYSTEMS™
45-D Accord Park, Norwell, MA 02061
617-871-5435

CIRCLE 60 ON READER SERVICE CARD

C Helper™

FIRST-AID FOR C PROGRAMS

Save time and frustration when analyzing and manipulating C programs. Use C HELPER's UNIX-like utilities which include:

DIFF and **CMP** – for "intelligent" file comparisons.
XREF – cross references variables by function and line.
C Flow Chart – shows what functions call each other.
C Beautifier – make source more regular and readable.
GREP – search for sophisticated patterns in text.

There are several other utilities that help with converting from one C compiler to another and with printing programs.

C Helper is written in portable C and includes both full source code and executable files for $135 for MS-DOS, CPM-80 or CPM-86. Use VISA, Master Card or COD.

Call: **617-659-1571**

Solution Systems™
335-L Washington Street
Norwell, MA 02061

CIRCLE 61 ON READER SERVICE CARD

The Evolution of ZCPR

PART II

By Richard Conn

Part I of this article, which appeared in the October issue of *COMPUTER LANGUAGE*, discussed the major concepts behind the evolution of ZCPR3 and compared CP/M and ZCPR3 memory maps. Part II will cover the ZCPR3 toolset, ZCPR3 shells, and sources to turn to for more information.

ZCPR3 toolset

A large part of the ZCPR3 system is a set of programs from which the user can create software development and applications environments. The ZCPR3 toolset has several classes of tools:

- Utilities, which perform basic functions such as erasing files and displaying directories
- Documentation, which supports the documentation of the system for the user, including an on-line reference aid (HELP facility)
- Programmer aids, which assist the software designer and programmer in debugging software
- Shells, which act as front ends to the ZCPR3 command processor and provide a different type of interface (such as menus) between the user and the ZCPR3 system
- Command file processors, which support the processing of files containing commands.

The ZCPR3 system contains over 70 programs in the form of .COM files and well over 100 commands in various forms (.COM files, FCPs, RCPs, etc.). For the most part, these tools are consistent in their syntax and use:

- When it is reasonable to accept one or more lists of files in conjunction with the function performed by a command, such is permitted, the *PRINT* command is of the form,

```
PRINT afn1,afn2,afn3, . . . options
```

(where each "afn" is an ambiguous file reference, like *.TXT, or an unambiguous form, like MYFILE.DOC)

- Most tools implemented as .COM files provide built-in documentation to remind the user in a brief format how to use the command and what its syntax is
- All tools in the toolset are documented in 200K-plus of on-line documentation (*.HLP) files, and the user can quickly index into these files and look up page after page of information on every tool in the toolset
- All tools use the Environment Descriptor as required to obtain the extended information they need to perform their functions, such as *PRINT*, which needs to know the size of the printer page—this is why ZCPR3 tools cannot normally run under "vanilla" CP/M.

Shells

Many features of the ZCPR3 system require an involved explanation, and the number of pages in this magazine would not be adequate space to explain all of ZCPR3. A ZCPR3 book, now being published, covers all of them in detail, but to give you a feeling for the magnitude of some of these features, let's focus in some detail on one subset of the ZCPR3 toolkit: shells.

The concept of the shell was incorporated as an integral part of the design of the ZCPR3 system. Having designed two shells under UNIX previously, I saw much value in the shell concept and wanted to implement it in a single-tasking environment like ZCPR3 so I could use it with my CP/M 2.2 programs.

A shell as defined for ZCPR3 is a program that runs in place of a command line interpreter (CLI) as a user interface to the operating system.

Under CP/M, the CLI in the CCP shows its execution by prompting the user with the "d>" prompt and accepting the user's command line. The CLI in the ZCPR3 CP does the same type of thing that the CLI in the CP/M CCP does, but the ZCPR3 CP always checks for the presence of a shell and runs it instead of the CLI if it sees that a shell has been specified.

Generally speaking, a shell can be implemented in two basic ways:

- As a program that executes its function and then terminates, only to be reinvoked later by the operating system before the CLI is invoked
- As a program that executes as a process in a multitasking operating system, so that rather than terminating between invocations, the shell process is suspended.

ZCPR3, having CP/M 2.2 compatibility as one of its main design criteria, implements shells as programs that are terminated (as in the first way), while shells written under UNIX are usually implemented as processes that execute commands as subprocesses and suspend themselves between invocations.

In implementing a UNIX-like shell, where process suspension is supported by the operating system, the shell can be executed as a program from the default shell (such as the Bourne Shell or the Berkeley Shell).

The default shell itself becomes a suspended process, and the new shell provides its CLI-replacement interface and executes commands by spawning them as subprocesses under itself, suspends itself (allowing these subprocesses to run), and resumes operation at the point of suspension when the subprocesses complete, having been passed a return code from the spawned subprocesses.

The complete state of the shell is preserved by the operating system without the shell having to be concerned with the details of maintaining its state at the point of subprocess invocation. Using the standard library functions provided under the /USR/INCLUDE directory in UNIX, process spawning and suspension become trivial problems since the library provides routines to perform these functions for the software designer.

In implementing a ZCPR3-based shell, the shell is re-executed as a program by the ZCPR3 CP, where processes are terminated and then later reinvoked. A shell stack is employed which allows the nesting of shells and acts as a message buffer so that the state of the shell can be recorded in a limited fashion and restored when the shell is reinvoked.

It is the responsibility of the shell to record and then restore its state. Other message buffers are also provided under the ZCPR3 system to record shell-specific information, but using the shell stack as a recording mechanism insures that state values set by a shell are not affected (necessarily) when a second shell is invoked by the first shell.

When the ZCPR3 CP invokes a program as a shell, a message is set to indicate this action, thereby letting the shell know that valid data on its previous state is recorded in the appropriate message buffers and it can access these buffers to restore the previous state.

Message buffers are also provided to allow programs to communicate with other programs which are executed at a later time and with a shell when it is reinvoked. This provides the shell with information such as the success code associated with a program's execution.

A variety of shells have been implemented under the ZCPR3 system. Since the system's Environment Descriptor includes a definition of the capabilities of the user's terminal, most of these shells are screen-oriented, issuing screen-oriented command sequences to a package of library routines which are translated into terminal-specific byte sequences for interpretation by the user's terminal.

Shells under ZCPR3 range in function from menu display-oriented command processors to file manipulation utilities with built-in menu facilities to dynamic debugging facilities that can examine the state of the computer's memory after a program being debugged has executed.

In addition, a shell definition program is provided in the ZCPR3 toolset so that any conventional command sequence—such as a word processor invocation followed by a compiler invocation on the file that was just edited—can be installed as a shell. A variety of shells and tools that control shells and shell stacks is provided under the ZCPR3 system, and full CP/M 2.2 compatibility is retained at all times.

Shell implementations, then, are made in two different schemes under ZCPR3 and UNIX. The UNIX scheme is the simplest and most complete from the shell's point of view—the entire state of the shell is preserved by the fact that it is being suspended by the operating system. The same state is restored once the subprocesses have terminated.

The ZCPR3 scheme is more complex and more incomplete from the shell's point of view—each shell must contain the code required to save and then restore its state to a reasonable degree of detail between invocations. The ZCPR3 library, Z3LIB, provides routines that simplify this process from the software designer's point of view, but state preservation and restoration must still be taken into account in the design of the shell.

The ZCPR3 technique of shell implementation provides some interesting additional potential not found in the UNIX technique. Since ZCPR3 is a single-tasking system and the shell state (and shell stack) are provided as messages, a tool that is executed under a shell may modify the shell state, terminate the shell, or invoke a new parent shell during the tool's execution.

The UNIX operating system protects a suspended process from being modified by another process, so modification of the shell state by a subprocess is not possible under UNIX. The subprocess can only return a result code or store a message in the form of a file for the shell to read when it is reinvoked.

Also, I am not aware of any technique that a subprocess can use under UNIX to terminate its parent process or switch to a different parent process during the child's execution. These capabilities have been found useful in several applications of shells under ZCPR3.

Both techniques are useful, having their own sets of benefits from an implementation point of view. From the perspective of a shell programmer—a person who programs environments into the language of the shell (such as the case when the shell is a menu shell)—the design techniques are different with these two implementations.

From the perspective of the user—the simplification of his or her perspective being the principal reason for implementing shells—the higher level of abstraction and greater ease of use of a system are virtually identical under ZCPR3 and UNIX.

Sources of information

Many bulletin boards around the world are carrying the ZCPR3 system in various forms, and the Special Interest Group in Microcomputers (SIG/M) of the Amateur Computer Group of New Jersey is a central, public-oriented distribution point for it. Several computer clubs are also offering it to their members. For those who are interested in acquiring ZCPR3 through SIG/M, contact SIG/M, P.O. Box 97, Iselin, N.J. 08830.

The first phase of the ZCPR3 distribution is contained on 14 8-in. IBM 3740 disks (241K each) and includes source code to the entire system. This distribution includes 50 .COM files, a large number of .HLP files for on-line documentation, the main system segments, SYSLIB3 (the software components library used to create the .COM files), and the installation manual and user's perspective.

The second phase of the ZCPR3 distribution was released in September and covered five more disks.

The ZCPR3 system is supported by Echelon Inc., which provides the following services to the ZCPR3 user community:

- A central distribution point for ZCPR3. It can be purchased there, and commercial firms may obtain licenses to use it. The basic purchase price (without source code) is $39, and a variety of additional options are available, bringing the price of all 14 disks to under $200. A self-installing version of ZCPR3 is being marketed by Echelon for $149 (installation without this can take up to four hours or more and requires knowledge of assembly language programming).
- Hardcopy of the installation manual and user's perspective document. Echelon will be selling a ZCPR3 book (500 pages long in draft form), which will be out before the end of the year at an expected price of $20.
- A bulletin board service through which information is distributed and users can report problems
- A user support service that is accessible by telephone
- A newsletter
- A marketing agent and service for those users who wish to write commercial ZCPR3 programs and sell them
- A feedback mechanism to the author of ZCPR3 (Richard Conn).

For more information on ZCPR3, write or call: Echelon Inc., 101 First Street, Los Altos, Calif. 94022, (415) 948-3820 or (415) 948-5321.

Richard Conn has a B.S. and M.S. in computer science. His current interests include operating systems, C and UNIX, and the Ada programming language.

The Preferred C Compiler

"...C86 was the only compiler we tested that ran every benchmark we tried and gave the expected results... Computer Innovations C86 was the compiler that our staff programmers used both before and six months after we conducted the tests."
J. Houston, BYTE MAGAZINE - February 1984

•FAST EXECUTION - of your programs.

•FULL & STANDARD IMPLEMENTATION OF C - includes all the features described by K & R. It works with the standard MSDOS Linker and Assembler; many programs written under UNIX can often be compiled with no changes.

•8087 IN-LINE - highly optimized code provides 8087 performance about as fast as possible.

•POWERFUL OPTIONS - include DOS2 and DOS1 support and interfaces; graphics interface capability; object code; and librarian.

•FULL LIBRARY WITH SOURCE - 6 source libraries with full source code the "large" and "small" models, software and 8087 floating point, DOS2 and DOSALL.

•FULL RANGE OF SUPPORT PRODUCTS FROM COMPUTER INNOVATIONS - including Halo Graphics, Phact File Management, Panel Screen Management, C Helper Utilities and our newest C_to_dBase development tool.

•HIGH RELIABILITY - time proven through thousands of users.

•DIRECT TECHNICAL SUPPORT - from 9 a.m. to 6 p.m.

Join The Professional Programmers Who Agree C86™ Is The C Compiler Of Choice

For Further Information Or To Order Call:
800-922-0169
Technical Support: (201) 542-5920

980 Shrewsbury Avenue
Suite PW509
Tinton Falls, NJ 07724

Computer Innovations, Inc.

C86™

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
UNIX IS A TRADEMARK OF BELL LABS. C86 IS A TRADEMARK OF COMPUTER INNOVATIONS, INC. MSDOS IS A TRADEMARK OF MICROSOFT.
PCDOS IS A TRADEMARK OF INTERNATIONAL BUSINESS MACHINES.

CIRCLE 11 ON READER SERVICE CARD

We Do
Windows!

OPTIONAL
8087
SUPPORT

FORGET

EVERYTHING YOU THOUGHT YOU KNEW ABOUT PROGRAMMING IN BASIC.

introducing:

Better BASIC™

BetterBASIC offers:

- Support of large memory (to 640K).
- Extensibility (Make your own BASIC!!)
- Speed. Sieve of Erastosthenes Benchmark:
  - **BetterBASIC**: 31.9 seconds.
  - IBM PC BASIC: 191.1 seconds.
- Program Block Structures.
- User defined Procedures and Functions.
- Local and Global Variables.
- Shared Variables.
- Recursion.
- Argument type validation.
- Optional arguments.
- Arguments passed by-value or by-address.
- Separately compiled program Modules.
- Simple interface to Assembly Language Procedures.
- Support for OEM hardware through extensibility.
- Useful set of Data Types:
  - Byte, Integer
  - Real (variable precision BCD) Ideal for business math.
  - String (up to 32768 characters)
  - Record Variables & Structures.
  - N-dimensional Arrays of any type.
  - Arrays of Arrays.
  - Pointer (of any type)

"It combines the best points of interpreted Basic, Pascal, Forth and Assembler... It's the first piece of software I'd spend my own money on." Susan Glinert-Cole
Technical Editor
PC Tech Journal

We are so sure you will like **BetterBASIC**, we will give you a 30-day money-back guarantee. Order **BetterBASIC** now!
BetterBASIC: $199.00
8087 Module: $99.00
Not convinced? Then try the **BetterBASIC** Sample and you will find that **BetterBASIC** is truly a major breakthrough in computer programming.
Sample disk: $10.00

General Information:

- Interactive programming language based on an incremental compiler.
- Syntax checked immediately on entry, with concise error reporting.
- Built-in Screen Editor allows on-line editing.
- Full IBM Graphics/Communications Support.
- Built-in Linker for separately compiled program Modules.
- Built-in Cross Reference Lister
- Built-in WINDOWS support!!
- 8087 math support

Computer Requirements:

- IBM PC, IBM PC/XT or compatible.
- PC/DOS 1.1, 2.0, 2.1
- 192K to 640K memory
- Usable on plain MS-DOS machines with reduced functionality.
  (no Editor, Graphics or Windows)

OEM & Dealer Inquiries Invited.

BetterBASIC is a trademark of Summit Software Technology, Inc.
IBM PC, IBM PC/XT and PC/DOS are trademarks of International Business Machines Corp.
MS-DOS is a trademark of Microsoft Corp.

CALL YOUR DEALER OR SUMMIT SOFTWARE AT 617-235-0729

Summit Software Technology Inc. ™ P.O. Box 99 Babson Park Wellesley, MA 02157

MasterCharge, Visa, P.O., Checks, Money Orders and C.O.D. accepted

CIRCLE 62 ON READER SERVICE CARD

UNIX SOFTWARE

UniPress Product UPDATE

LATTICE® C NATIVE AND CROSS COMPILERS FOR THE 8086

AMSTERDAM COMPILER KIT

Outstanding software development tools

Lattice C Cross Compiler to the IBM-PC

- Highly regarded compiler producing fastest and tightest code for the 8086 family.
- Use your VAX or other UNIX machine to create standard Intel object code for your 8086 (IBM-PC)
- Full C language and standard library, compatible with Unix.
- Small, medium, compact and large address models available.
- Includes compiler, linker, librarian and disassembler.
- 8087 floating point support.
- MS-DOS 2.0 libraries included.
- Send and Receive communication package optionally available to communicate between Unix and MS-DOS.

Prices:	Hosted On	
	VAX/Unix and VMS	$5000
	MC68000/8086	3000
	Send and Receive	500

Lattice C Native Compiler for the 8086

- Runs on the IBM-PC under MS-DOS 1.0 or 2.0.
- Produces highly optimized code
- Small, medium, compact and large address models available.
- Compiler is running on thousands of 8086 systems.

Price: $425

Plink (Optional) for use with native Lattice

- Full function linkage editor including overlay support.

Price: $395

Amsterdam Compiler Kit

- Package of compilers, cross compilers and assemblers.
- Full C and Pascal language.
- Generates code for VAX, PDP-11, MC68000, 8086 and NSC16000.
- Hosted on many Unix machines.
- Extensive optimization.

Price:		
	Full system — source	$9950
	Educational Institution	995

OEM terms available • Much more Unix software, too! • Call or write for more information.

Mastercard and Visa

UniPress Software, Inc.
2025 Lincoln Highway, Edison, NJ 08817
201-985-8000 • Order Desk: 800-222-0550 (outside NJ) • Telex 709418

Lattice is a registered trademark of Lattice, Inc. Unix is a trademark of Bell Laboratories.
MS-DOS is a trademark of Microsoft.

UNIX SOFTWARE

CIRCLE 38 ON READER SERVICE CARD

Learn to Think in Ada

By Do-While Jones

The Dept. of Defense is encouraging the use of Ada because it expects to reduce software costs by the selection and consistent use of this particular programming language. It is anticipated that at least part of the savings will occur because programs written in Ada will have fewer bugs and be easier to maintain.

Can simply changing to Ada really improve your own program? Yes it can, but in order to write good Ada programs you have to learn to *think* in Ada.

Listing 1 shows two solutions to exercise 3.4B in S. J. Young's book, *An Introduction to Ada*. The first solution is written in BASIC and the second one in Ada. Can you guess what the question was just by looking at these two answers?

Suppose that the statements in Listing 1 are part of a program that you have to maintain. Do you know what those lines of code are supposed to do? Can you be sure that you can change them without creating a new problem?

Programs that are easy to understand are said to be "maintainable" because a competent programmer can make changes to the program (either to fix a programming error or add a new feature) without introducing errors. The DOD expects Ada programs to be more maintainable than programs written in any other language.

Is the program in Listing 1B any easier to understand than Listing 1A? I don't think so. Listing 1B tells us that *I* and *J* should be integers, but that's about the only difference I see in the two programs.

Now look at Listing 2. I'll bet you can figure it out after reading it through just once. Both Listings 1B and 2 were written in Ada, but Listing 2 is much easier to understand.

The problem with Listing 1B is that it wasn't really written in Ada. The student who wrote that program was probably thinking of a BASIC or a FORTRAN solution and simply translated that solution into Ada.

```
10  INPUT "WHAT IS THE FIRST NUMBER";I
20  INPUT "WHAT IS THE SECOND NUMBER";J
30  K=(I+1) MOD 2 + J MOD 2
40  PRINT "THE ANSWER IS ";
50  IF K = 0 THEN PRINT "TRUE." : GO TO 70
60  PRINT "FALSE"
70  END
```

Listing 1A.

```
with TEXT_IO; use TEXT_IO;
procedure EX_3_4_B is
  package INT_IO is new INTEGER_IO(integer);
  use INT_IO;
  I, J, K : integer
begin
  put ("WHAT IS THE FIRST NUMBER? "); get(I); new_line;
  put ("WHAT IS THE SECOND NUMBER? "); get(j); new_line;
  K := (I+1) mod 2 + J mod 2;
  put("THE ANSWER IS ");
  if K = 0
    then put("TRUE");
    else put("FALSE");
  end if;
  new_line
end EX_3_4_B;
```

Listing 1B.

Ada programs shouldn't be written the same way that BASIC programs are. BASIC forces programmers to get into the habit of thinking of a sequential solution to the problem and then writing the solution as a series of steps in a particular order. If the programmer discovers that something was left out, he or she has to renumber the program and insert the steps where they belong or add the steps to the end and change the program flow using *GOTO*s.

Ada doesn't have to be written from beginning to end, and it is easier not to try. In fact, the last line in an Ada program is always the third line I write. That's because it is easier to write an Ada program from the top to the bottom rather than from the beginning to the end. Here's how I wrote the program in Listing 2. First I wrote the skeleton:

```
procedure Solve_Exercise_3_4_B is
begin
end Solve_Exercise_3_4_B;
```

Next I wrote a sequence of statements expressing a general solution to the problem and put it between the *begin* and *end* statements (Listing 3).

Even if you have never seen an Ada program before, you probably figured out that *put* in Ada is just like *PRINT* in BASIC, and *get* is the equivalent of *INPUT*. The semicolons mark the end of a statement just like the colon does in Microsoft BASIC. Ada ignores carriage returns and line feeds, so you can put multiple statements on one line or break one statement into several lines if it makes the meaning clearer.

I chose to put three statements on one line because those three statements go together logically. The *put* prompts the user, the *get* gets the user's response, and *new_line* sends a carriage return and line feed to the CRT screen after the user has entered the data.

put, *get*, and *new_line* are procedures I use often. It would be wasteful to rewrite them and recompile them every time I needed them. They are all in a general purpose I/O package called TEXT_IO, but the *put* procedures for integers and Booleans are generic and have to be "instantiated" before they are used. (Listing 1B shows how the INTEGER_IO package is instantiated.)

I find it a nuisance to instantiate INTEGER_I/O every time I need it, so I have written my own I/O package called CONSOLE_IO, which contains all the IO routines I need, ready to run. I can tell Ada to use these routines by placing a "context clause" at the beginning of the program:

```
with CONSOLE_IO; use
     CONSOLE_IO;
```

CONSOLE_IO has *get* procedures for characters, strings, integers, and Boolean variables. I had to tell Ada (and CONSOLE_IO) to treat *I* and *J* as integers, not character strings:

```
I, J : integer:
```

I wrote the solution to the problem without worrying about how I was going to do it. I just wrote the answer I wanted:

```
TEST_RESULT := I_IS_ODD and
     J_IS_EVEN;
```

If you've never seen an Ada assignment statement, the colon might have confused you. Ada has two kinds of equal signs. The assignment operator (:=) means "Make it equal." The equality test (=) means "Is it equal?"

TEST_RESULT, *I_IS_ODD*, and *J_IS_EVEN* are Boolean variables. They can have values of TRUE or FALSE. *TEST_RESULT* will be TRUE if *I_IS_ODD* is TRUE and *J_IS_EVEN* is TRUE. Ada needs to know what kind of variables these are, so they are declared near the beginning of the program:

```
TEST_RESULT, I_IS_ODD,
     J_IS_EVEN : Boolean;
```

J_IS_EVEN may seem like a long name for a variable. Ada variable names are theoretically limited to the number of characters that fit on one line, and all the characters (including the underscores) are significant. In practice, variable names are limited by the programmer's desire to type as little as possible.

When I wrote the solution the problem automatically broke itself down from one problem to two smaller problems. Then I had to figure out if *I* was odd and if *J* was even. Those were easy problems to solve. The modulus operator (mod) tells if a number is evenly divisible by two:

```
if I mod 2 = 1
     then I_IS_ODD := TRUE;
     else I_IS_ODD := FALSE;
end if;
if J mod 2 = 0
     then J_IS_EVEN := TRUE;
     else J_IS_EVEN := FALSE;
end if;
```

The program in Listing 2 is longer than the program in Listing 1A if you measure length by number of lines. If you measure length by the time it took to write the program it was probably shorter. The statement,

```
if I mod 2 = 1 then I_IS_ODD :=
     TRUE;
```

is a simple definition that required no great mental effort to conceive. The calculation,

```
with CONSOLE_IO; use CONSOLE_IO;
procedure Solve_Exercise_3_4_B is
  I, J : integer;
  TEST_RESULT, I_IS_ODD, J_IS_EVEN : boolean;
begin
  put ("Please enter an integer ""I"" "); get(I); new_line;
  put ("Now enter another integer ""J"" "); get(J); new_line;
  if I mod 2 = 1
    then I_IS_ODD := TRUE;
    else I_IS_ODD := FALSE;
  end if;
  TEST_RESULT :I_IS_ODD and J_IS_EVEN;
  put("the test turned out "); put (TEST_RESULT); new_line;
end Solve_Exercise_3_4_B;
```

Listing 2.

```
put("Please enter an integer ""I"" "); get(I); new_line;

put("Now enter another integer ""J"" "); get(J); new_line;

TEST_RESULT := I_IS_ODD and J_IS_EVEN;

put("The test turned out "); put(TEST_RESULT); new_line:
```

Listing 3.

```
K=(I+1) MOD 2 + J MOD 2
```

required several thought processes. First the student had to realize that *K=0* should represent TRUE. Then he had to realize that he wanted to use the + operator to do the *AND* function. Then the programmer had to figure out how to get an odd number to yield a zero result.

In the process, the student might have considered using the * operator (which is usually used for logical *AND*) but couldn't figure out how to get a zero result with it. If he had thought to let K=1 represent TRUE, he could have done it. (That's left as an exercise for the reader.) It probably took more time to write the one line equation for *K* than it did to write both *IF* statements in Listing 2.

Early in this article, I was careful to say that Listing 1 shows two solutions to the problem. I never said they were correct solutions. Look at them carefully. Will they work for all cases? (Hint: Try *I* = −6 and *J* = 5.)

You might argue that the program in Listing 1A could be improved by adding remarks. Of course, you are right. It would help a great deal to add the following statements:

```
1 REM - THE USER MUST ENTER
    TWO INTEGERS, I AND J.
2 REM - THE PROGRAM THEN TESTS
    TO SEE IF I IS ODD AND J IS
    EVEN.
3 REM - THE TEST RESULTS ARE
    PRINTED ON THE SCREEN.
```

This relieves the maintenance programmer of figuring out what the program is supposed to do. But that is only part of the problem. The maintenance programmer still has to figure out how it works. That can be difficult when the logic is tricky.

By the way, the programs in Listing 1 are correct. I tried to bluff you into thinking that −5 mod 2 was −1, so *I* = −6 and *J* =5 would yield −1 + 1 = 0, which would cause the program to print TRUE (which is the wrong answer).

Maybe I confused you, and maybe I didn't, but if you had to think twice, I made my point. It is hard to follow the logic in Listing 1, but the logic in Listing 2 is straightforward. That makes Listing 2 easier to maintain.

Will programs written in Ada have fewer bugs and be easier to maintain? They will if they are written like Listing 2. Unfortunately, anyone who can write a bad program in any other programming language can also write a bad Ada program.

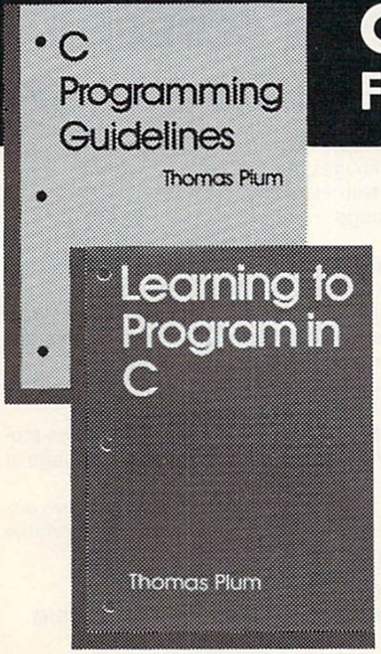
Ada isn't a miracle cure. Ada is simply a language that makes it easy to write good programs if the programmer can break away from constraining his or her mind with old program limitations. The student who wrote program 1B didn't use Boolean variables because he was used to using a language limited to integers, real numbers, and strings.

Of course Ada has other advantages that will lower software costs. User-defined enumeration types, packages, tasking, exceptions, information hiding, and separate compilation all make programming easier. In future articles I hope to share some of these advantages with you.

Do-While Jones graduated from the Univ. of Nebraska in 1971. Since that time he has been designing weapons.

C LANGUAGE PROGRAMMING

From Plum Hall...the experts in C training



Learning to Program in C 372 pp., 7½" x 10", Price $25.00
A practical, step-by-step guide for everyone acquainted with computers who wants to master this powerful "implementer's language" Inside, you will learn how to write portable programs for the full spectrum of processors, micro, mini and mainframe

C Programming Guidelines 140 pp., 7½" x 10", Price $25.00
A compilation of standards for consistant style and usage of C language. Arranged in manual page format for easy reference, it presents time-tested rules for program readability and portability.

FREE
C LANGUAGE POCKET GUIDE!

A handy C language programming pocket guide is yours *free* when you order either (or both) of the manuals above. A full 14 pages of valuable C language information!

PLUM HALL
1 Spruce Av, Cardiff NJ 08232
The experts in C and UNIX™ training.
Phone orders: 609-927-3770

Please send me:
_____ information on C and UNIX Training Seminars
_____ copies of Learning to Program in C @ $25.00/copy
_____ copies of C Programming Guidelines @ $25.00/copy
NJ residents add 6% sales tax.

NAME ____________
COMPANY ____________
ADDRESS ____________
CITY/STATE/ZIP ____________
☐ Check ☐ American Express ☐ Master Card ☐ Visa
CARD # ________ EXP. DATE ____/____ Signature ________

UNIX is a trademark of AT&T Bell Laboratories

CIRCLE 50 ON READER SERVICE CARD

"C/80 . . . the best software buy in America!"

—MICROSYSTEMS

Other technically respected publications like *Byte* and *Dr. Dobb's* have similar praise for **The Software Toolworks' $49.95** full featured **'C' compiler** for CP/M® and HDOS with:

- I/O redirection
- command line expansion
- execution trace and profile
- initializers
- Macro-80 compatability
- ROMable code
- and much more!

"We bought and evaluated over $1500 worth of 'C' compilers . . . C/80 is the one we use."

— Dr. Bruce E. Wampler
Aspen Software
author of *"Grammatik"*

In reviews published worldwide the amazing **$49.95 C/80** from **The Software Toolworks** has consistently scored at or near the top — even when compared with compilers costing ten times as much!

The optional **C/80 MATHPAK** adds 32-bit floats and longs to the C/80 3.0 compiler. Includes I/O and transcendental function library all for only **$29.95!**

C/80 is only one of 41 great programs each **under sixty bucks.** Includes: LISP, Ratfor, assemblers and over 30 other CP/M® and MSDOS programs.

For your **free** catalog contact:
The Software Toolworks
15233 Ventura Blvd., Suite 1118,
Sherman Oaks, CA 91403 or call 818/986-4885 today!

CP/M is a registered trademark of Digital Research.

CIRCLE 26 ON READER SERVICE CARD

Fortran Scientific Subroutine Package

Contains Approx. 100 Fortran Subroutines Covering:

1. Matrix Storage and Operations
2. Correlation and Regression
3. Design Analysis
4. Discriminant Analysis
5. Factor Analysis
6. Eigen Analysis
7. Time Series
8. Nonparametric Statistics
9. Distribution Functions
10. Linear Analysis
11. Polynomial Solutions
12. Data Screening

Sources Included. Microsoft 3.2 compatible.

$295.00

FORLIB-PLUS™

Contains three assembly coded LIBRARIES plus support, FORTRAN coded subroutines and DEMO programs.

The three LIBRARIES contain support for GRAPHICS, COMMUNICATION, and FILE HANDLING/DISK SUPPORT. An additional feature within the graphics library is the capability of one fortran program calling another and passing data to it. Within the communication library, there are routines which will permit interrupt driven, buffered data to be received. With this capability, 9600 BAUD communication is possible. The file handling library contains all the required software to be DOS 3.0 PATHNAME compatible.

$69.95

Strings & Things ™
Character Manipulation and Much More!
$69.95

ALPHA COMPUTER SERVICE
P.O. Bos 2517
Cypress,CA 90630 **(714) 894-6808**

California residents, please add 6% sales tax

CIRCLE 2 ON READER SERVICE CARD

DESIGNER SCREENS

"A 100 to 1 Productivity
Increase Over Coding"

Provides full-screen editing of terminal screen design images. And, a linker that generates self-relocating, 8080 machine language, run-time support.

Makes it easy to implement on-screen forms, menus, help screens, boiler-plate notices, and even simple animation.

Run-time support for input includes: data type control, decimal alignment, a type ahead buffer, end-user edit commands, and everybody's favorite, "Fred's Magic Window."

Fred's Magic Window can display field-by-field input instructions as needed, automatically.

Can be used with any computer language that allows programmed calls to CP/M 2.2. Great with assembly language or BDS C.

Runs on 80 × 24 or larger ASCII terminals. Supports five display attributes and line drawing. Designs are transportable between installed terminals.

Manual only:	**$ 10.00**	**(Check it out!)**
Software:	**185.00**	**(Supplied on: 8" SSSD CP/M or call.)**
Complete:	**$195.00**	

(Calif. residents add sales tax)

Austin E. Bryant Consulting
P.O. Box 1382, Lafayette, CA 94549
(415) 945-7911

MasterCard VISA

CP/M is a trade mark of Digital Research
BDS C is a trade mark of BD Software

CIRCLE 7 ON READER SERVICE CARD

PUBLIC DOMAIN SOFTWARE REVIEW

By Tim Parker

This month we're going to begin by filling in some gaps for the 16 bit-ers.

Specifically, we'll concentrate on the Big Blue machine and its clones, compatibles, look-alikes, etc. (Other machines will follow, though I'm still waiting for a Sage to appear on my desk!)

The IBM PC has brought a lot of new owners to the personal computer market, but I have not found one machine in the past few years that has generated more of a love-hate relationship than this one. There is a lot good and a few major annoyances. But then, I'm not here to voice my opinions of the IBM, just to tell you what you can run on it. (But *why* the silly keyboard?)

As we concentrate on public domain software for the IBM PC, it must be pointed out that there is one minor problem that crops up occasionally. Naturally, as the programs were developed on the IBM, they were written under PC-DOS. Some are in Version 1.1, while others are 2.0. And therein lies the crunch.

In many cases, the DOS 1.1 stuff will not run on DOS 2.0 and vice versa. Most do but be warned! Generally, the early stuff is the most suspect as the existence of DOS 2.0 was not even contemplated at the time.

As for compatibility, most of the software will run fine on the look-alikes as long as they maintain some standard of IBM compatibility in their structure. Some programs require screen addressing or memory addressing that the compatibles don't all have.

Unfortunately, there's no way to be certain whether a program will work or not until it's tried. The best guess one can make is that if the machine is a close clone (such as the Compaq) then it probably will work, while more distant clones that have changed screens or other essentials (whether for better or worse) may not.

Finally, as these programs were mostly developed on IBMs, they are all PC-DOS. Whether they work under MS-DOS depends on the way the program is set up. Again, that is unpredictable to any great extent. And so, with that preamble out of the way, we'll take the first of a number of quick walks through the IBM PC public domain software library.

IBM public domain software can be located in several places scattered throughout the country. One organization that has impressed me by its standards of organization is the PC Software Interest Group (PC-SIG). (Its address is at the end of the column.)

PC-SIG publishes a smallish book that lists the contents of the public domain library it handles. Last time I checked, it was almost at 200 disks, so it has probably exceeded that by now. PC-SIG's disks are very reasonably priced at $6.00 each, with a $4.00 postage and handling surcharge ($10.00 outside the U.S.), and they offer telephone support for service and ordering.

PC-SIG supports both the true public domain software and the newer "user-supported program" concept. Simply stated, a user-supported program is one that the authors have released for general distribution through the usual channels such as public domain houses, bulletin boards, etc., but they expect a donation in return from the user if the software is appreciated.

The donations are usually between $10.00 and $30.00. Naturally, there is no obligation on the part of the user to pay anything, but it is hoped that since many of these programs are of extremely good quality, there will be a streak of generosity in the user to recompense the author for the effort.

The idea of user-supported software is not really new. Programs with a tag that asked for donations if deserved have been around as long as public domain software has existed, but this seems to be an organized effort to target the IBM PC market for a structured examination of the concept.

Some of the software available through this scheme is very good indeed. Generally, it is above the usual quality of public domain "freebie" software and on par with the commercial material (although there is quite a range in the quality from some of the authors).

The beauty of the idea, of course, is the "try it before you buy it" aspect. If the software is worthwhile then there will probably be little reluctance on the part of the user to pay out $20.00 or $30.00 compared to the hundreds of dollars that have to be laid out in advance with no guarantee for commercial products. It will be interesting to see how the PC SIG organization of this concept proceeds.

Back to business. PC software: where to begin? Probably with a few of the more useful utility disks and then a quick jump around some of the volumes. Some of the programs mentioned have been covered in greater detail in previous columns, and so they should be referred to when necessary. While the 8-bit version may have been discussed specifically, there are minor (if any) changes to the PC/MS-DOS environment. (If anything major is changed, it will be pointed out.)

Most of the programs mentioned are executable directly, either as .COM or .EXE files. Some are .BAS files and require the standard BASIC program to run them. Many of the files are supplied as .BAT (BATch files) and are thus listable on the console or other output devices.

PC-SIG Disk 185 contains a version of DD (a directory sort program) along with other directory programs such as DIR2, SDIR, and CATALOG. This version of DD, however, sorts by date, and so is not the same program as the 8-bit DD.COM discussed last month.

The assembler source for one of the directory programs is included for customization, examination, or modification as required. Included is a squeezer-unsqueezer pair called SQIBM/USQIBM and ZSQ/ZUSQ, respectively, which are essentially the same as the SQ/USQ files discussed earlier. Both have documentation files included on the disk. A program called SEC&BYTE prints a table of the diskette sectors and bytes.

PC-SIG Disk 186 is a collection of CRT and Epson (IBM) printer utilities. A program called NOCOLOR switches the color graphics card to black and white mode quickly. Two scroller programs are included to provide screen scrolling control that the standard machine doesn't offer. One of them includes the source code (assembler). A print spooler program is included that can also be used for data communications.

Several Epson utility programs round out the disk to allow rapid setting of the Epson for different type fonts, spacing, graphics, lines per page, etc. These programs all seem to offer only one feature each. I wrote a menu program that can reside in high memory and allow toggling of the features all from one program, instead of loading a new program every time. (All those disk accesses take so many microseconds!) However, different strokes . . .

One disk cataloging program set can be found on PC-SIG Disk 106. This disk has a cataloging system that can be implemented with or without BASRUN. Documentation is available interactively or through a document file. Although the document is short at five pages, it seems to cover all that is required. The usual cataloging features are available: complete disk catalogs can be printed, specific files searched for and displayed, and all programs sorted as required.

The DISKCAT system worked with no problems at all, and although I have an avowed dislike of cataloging systems (for reasons that will not be gone in to here), this seems to do all that Ward Christensen's XCAT/NCAT system did for CP/M 80.

There are also about 14 disks crammed with utility programs pulled off bulletin boards and the like, all grouped loosely together under the "utility" title. As can be expected, there is the usual assortment of useless stuff but also a few goodies.

PC-SIG Disk 111 has a program called BIGCALC that functions as a 100-digit precision calculator. (Thanks, but my HP 41CV works just fine. But there is the possibility that my debts may overflow the precision! I'll keep this one on file just in case.) The usual calendar program is on the same disk and can print out any month/year as required. A couple of other programs that are of use round out the disk, but you begin to get the idea.

Enough of utilities. (Check PC-SIG's catalog for all the utility programs you'll ever need.) Now for the good stuff!

CHASM (cute name) stands for CHeap ASseMbler and is one of the aforementioned user-supported software programs. CHASM is an assembler written entirely in BASIC and includes a tutorial on the 8086 assembly language. It was interesting to look it over, and it does the job remarkably well. (The disk is available as usual from PC-SIG, as Disk 10. They suggest a donation of $20.00 from all users who are so inclined to donate.)

PC-SIG Disk 31 has the Mountain View Press public domain version of Forth. MVPFORTH is a good version of Forth (I can't say it's the best, but then I'm not sure what is) and for a first Forth it is very highly recommended. A documentation file is on the disk, but as with all Forths I've seen, the best bet is to go out and buy *Starting FORTH* by Leo Brodie.

MVPFORTH can be brought up either from a cold boot (usual) or from inside DOS (not so usual but nice to have). The Forth screens for MVPFORTH are on PC-SIG Disk 32. (Note that this is not a DOS disk and can't be copied by DOS! Errors will be reported at every step. The directory is on screen 11.)

An IBM version of the infamous and irreplaceable CP/M 80 XMODEM (AMODEM, MODEM7) is available on PC-SIG Disk 54 with copious notes about using IBM asynchronous communications.

Three disks are devoted to Pascal Tools (PC-SIG Disks 130, 131, and 132), with good documentation, manuals, source files, and other goodies. Most of the util-

THE MOST EXTENSIVE

THE GREENLEAF FUNCTIONS
Library for C Programmers
Total Access to IBM PC and XT
Compatible with DOS 2.0, 1.1, CI C86, Lattice, and Microsoft C – Versions 1 and 2

PARTIAL CONTENTS
• DOS 2.0 - over 25 functions • Complete Video Access for Text and Graphics •Over 60 String Functions • Rainbow Series Color Text • Time and Date • Over 40 Printer Functions • Function and Special Keys • RS232 Async • All BIOS Functions • Software Diagnostics • Disk functions • Utility functions • and more . . .

THE GREENLEAF FUNCTIONS . . .
Nearly 200 functions, 220 page manual, 3 Libraries, Extensive Examples of each function, Full Source Code

$175.00
Add $7.00 for shipping.
Specify Compiler
MC/VISA Accepted
Prices subject to change without notice.
Dealer Inquiries Welcome.
(214) 446-8641

C LIBRARY ANYWHERE
FOR THE IBM AND PC XT

GREENLEAF SOFTWARE©
GREENLEAF SOFTWARE, INC. • 2101 HICKORY DRIVE • CARROLLTON, TEXAS 75006

CIRCLE 29 ON READER SERVICE CARD

(LISP) FOR A.I.
UO-LISP Programming Environment
The Powerful Implementation of LISP
for MICRO COMPUTERS

LEARN LISP System (LLS.1) (see description below) $39.95
UO-LISP Programming Environment Base Line System (BLS.1) $49.95

Includes: Interpreter, Compiler, Structure Editor, Extended Numbers, Trace, Pretty Print, various Utilities, and Manual with Usage Examples. (BLS.1) expands to support full system and products described below.

UO-LISP Programming Environment: The Usual LISP Interpreter Functions, Data Types and Extensions, Structure & Screen Editors, Compiler, Optimizer, LISP & Assembly Code Intermixing, Compiled Code Library Loader, I/O Support, Macros, Debug Tools, Sort & Merge, On-Line Help, Other Utility Packages, Hardware and Operating System Access, Session Freeze and Restart, Manual with Examples expands to over 350 pages. Other UO-LISP products include: LISPTEX text formatter, LITTLE META translator writing system, RLISP high level language, NLARGE algebra system. Prices vary with configurations beyond (BLS.1) please *send for FREE catalog.*
LEARN LISP System (LLS.1): Complete with LISP Tutorial Guide, Editor Tutorial Guide, System Manual with Examples, Full LISP Interpreter, On-Line Help and other Utilities. LEARN LISP fundamentals and programming techniques rapidly and effectively. This system does not permit expansion to include the compiler and other products listed above.
LISP Tutorial Support (LTS.1): Includes LISP and Structure Editor Tutorial Guides, On-line Help, and History Loop. This option adds a valuable learning tool to the UO-LISP Programming Environment (BLS.1). *Order (LTS.1) for* $19.95.
REQUIRES: UO-LISP Products run on most Z80 computers with CP/M, TRSDOS or TRSDOS compatible operating systems. The 8086 version available soon.
TO ORDER: Send Name, Address, Phone No., Computer Type, Disk Format Type, Package Price, 6.5% Tax (CA residents only), Ship & Handle fee of $3.00 inside U.S. & CN, $10 outside U.S., Check, Money Order, VISA and MasterCard accepted. With Credit Card include exp. date. Other configurations and products are ordered thru our *FREE catalog.*

Northwest Computer Algorithms
P.O. Box 90995, Long Beach, CA 90809 (213) 426-1893

CIRCLE 46 ON READER SERVICE CARD

ities included are created by a batch file. One program allows access to the FCB (file control block) for those who like to get their feet right into their operating systems. Pascal is also supported on other disks in the library, such as Disk 36, which is filled with utilities.

A screen editor written in C is available on Disk 137. This is the same program as the CP/M 80 editor called ED—not Digital Research's ED.COM. (If there are any CP/M 80 readers still with me, these are available on SIG/M Volume 76, along with some utilities from Software Tools of Australia.) I can't say I'd take ED over WordStar or one of my other word processors, but I am very impressed with it as a programming effort, and some people use it as a primary programming editor.

XLISP (mentioned a month or two ago for CP/M 80 and CP/M 86 machines) is on PC-SIG Disk 148. I'm glad to see that this made the transition to PC/MS-DOS, and the transfer has been done with total integrity. Both source files (written in C) and documentation are included.

Finally, to wrap up the IBM stuff—boy are there ever a lot of games! Most are translations of the usual (ho hum) games from BASIC programs in the CP/M 80 world, but there are a few that would tend to be looked at more than once.

PC-CHESS is available on PC-SIG Disk 120. It plays a good game, although as with most chess programs, when the more advanced look-ahead features are required, it becomes fairly time consuming. It comes in two versions, one for two players and the other the more traditional player vs. computer. Both .EXE files are supplied in squeezed format (the unsqueezer is on the disk), which raised an eyebrow when first seen. The reason for squeezing executable code still remains unsolved to this observer!

Ignoring the mass of basic games from the current book literature, there are versions of Star Trek all over the place (example—PC-SIG Disk 178), as well as arcade-like games (PC-SIG Disk 177). Pacman (in a monochrome version) is on PC-SIG Disk 173 with an ESP testing program. No comment on the latter: if you believe in ESP you'll know what I feel!

As for the rest of the games, check the catalog.

So much for this month's IBM discourse. I'll cover more of the available public domain material for PC/MS-DOS in future columns. Meanwhile, on to the language I'd promised to look at. The language this month is PISTOL, partly because of the previously mentioned release of Forth-83 and also because of the apparent boom in stack-oriented languages.

PISTOL (Portably Implemented STack Oriented Language) was written by Ernest E. Bergmann of Lehigh Univ. In many ways it resembles Forth and other stack-oriented languages such as STOIC (on which it was modeled), but it stands alone because of many unique features.

Stack-oriented languages are somewhat difficult to explain quickly. Anyone who has used a Hewlett-Packard calculator is acquainted with RPN (Reverse Polish Notation), which is stack oriented. Essentially, any numbers input into the system are maintained on a pile that allows operations on the last one or two items entered. Stack manipulation is readily accomplished.

The overall effect of stack systems is a vastly increased computing power over standard implementations because of the speed and flexibility of the stack concept. Too much to delve into here . . . take my word for it: when one of these languages is used, it tends to become addictive and often replaces other languages. A great many operating systems have been written in stack-oriented languages.

Bergmann developed PISTOL instead

QUALITY SOFTWARE AT REASONABLE PRICES

CP/M Software by

Poor Person Software

Poor Person's Spooler **$49.95**

All the function of a hardware print buffer at a fraction of the cost. Keyboard control. Spools and prints simultaneously.

Poor Person's Spread Sheet **$29.95**

Flexible screen formats and BASIC-like language. Preprogrammed applications include Real Estate Evaluation.

Poor Person's Spelling Checker **$29.95**

Simple and fast! 33,000 word dictionary. Checks any CP/M text file.

aMAZEing Game **$29.95**

Arcade action for CP/M! Evade goblins and collect treasure.

Crossword Game **$39.95**

Teach spelling and build vocabulary. Fun and challenging.

Mailing Label Printer **$29.95**

Select and print labels in many formats.

Window System **$29.95**

Application control of independent virtual screens.

All products require 56k CP/M 2.2 and are available on 8" IBM and 5" Northstar formats, other 5" formats add $5 handling charge. California residents include sales tax.

Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

CIRCLE 51 ON READER SERVICE CARD

LOWER PROGRAMMING MAINTENANCE AND DEVELOPMENT COSTS

{SET:**SCIL**}

The Source Code Interactive Librarian *for microcomputers.*

- **SCIL** keeps a historical record of all changes made to the library.
- **SCIL** maintains any source code regardless of language, including user documentation and text material.
- **SCIL** allows software engineers to work with source code as they do now, *using any ASCII text editor.*
- **SCIL** saves disk space by storing only the changes made to the program.
- **SCIL** provides a labeling capability for ease of maintaining multiple versions and multiple releases.
- **SCIL** offers unlimited description in the program library directory.
- *High visibility displays* with varied intensity for ease of viewing insertions and deletions.
- **SCIL** is available on CP/M, MP/MII, MS-DOS, PC-DOS and TurboDOS.

{SET} Get {SET} for Success
{SET:**SCIL**™} is a product of System Engineering Tools, Inc.
645 Arroyo Drive, San Diego, CA 92103

Registered Trademarks: CP/M, MP/M II, Digital Research Inc.; MS-DOS, Microsoft Corp.; PC-DOS, IBM Corp.; TurboDOS, Software 2000, Inc.

For more information call (619) 692-9464.

CIRCLE 64 ON READER SERVICE CARD

BRIEF

For the *first* time, a programmer's editor that is both intuitive *and* powerful

...and configurable to suit your style

The New Standard. No longer does an Editor have to be "in your way" to provide full power. By combining power *with* natural flow, the new advanced BRIEF is in a class by itself.

BRIEF lets you concentrate on programming. Your thoughts flow smoothly, intuitively. 15 minutes is all you need to become fully productive. You can then do precisely what you want quickly, with minimum effort and without dull repetitions.

BRIEF adapts to your style. You can use BRIEF without modification, because it's distributed with an "ideal" configuration. Or you can make any change you want, add any feature of your own. Reconfigure the whole keyboard or just the Function Keys. Change the way the commands work or just the start-up defaults.

Availability: PCDOS-compatible systems with at least 192K and one floppy drive are required. Though your initial copy is protected, an unprotected version is available when you register BRIEF.

Pricing: Only $195 . . . with discounts for volume end-users. A demonstration version is available for only $10 and can be available towards any Solution Systems purchase.

Win $1,000 and substantial recognition for the Outstanding Practical BRIEF Macro. Other awards will also be given.

BRIEF'S PERFORMANCE IS NOT EQUALLED IN MICROS, MINIS AND MAINFRAMES

- Full UNDO (N TImes)
- Edit Multiple Large Files
- True Automatic Indent for C
- Exit to DOS Inside BRIEF
- Uses All Available Memory
- Intuitive Commands
- Tutorial
- Repeat Keystroke Sequences
- Windows (Tiled and "Pop Up")
- Unlimited File Size
- Reconfigurable Keyboard
- Online Help
- Search for Complex Patterns
- Mnemonic Key Assignments
- Horizontal Scrolling
- Comprehensive Error Recovery

PLUS a Complete, Powerful, Readable, Compiled MACRO Language

Try BRIEF. Use the Demo . . .
or the full product for 30 days.
Call or write us . . .
617-659-1571

Solution Systems™

BRIEF is a trademark of UnderWare.
Solution Systems is a trademark of Solution Systems.

335-L Washington St., Norwell, MA 02061

CIRCLE 27 ON READER SERVICE CARD

of settling for Forth or STOIC on the assumption that such languages should be transportable between mini- and microcomputers with minimal transposition problems. Thus, portability was a major design problem, especially when instruction sets and word lengths had to be considered. Also, as Bergmann points out in one of the accompanying document files, user friendliness was essential.

Many readers will undoubtedly ask "Why bother?" with another Forth-like language. PISTOL differs from Forth in a number of ways. Strings are as easily manipulated in PISTOL as numbers are and can be easily defined.

The prompt in PISTOL is different than the one in Forth. Like STOIC, it displays the number of elements in the parameter stack, the current number base in use, and the nesting depth, if applicable.

PISTOL lacks the interpretive mode that Forth has, but this is more of a blessing than a hindrance. Everything goes into a compile buffer, which simplifies learning the language and also the coding of the thing. Thus, for immediate execution of a statement, it does not have to be structured as a definition. Along the same lines, the disassembler, editor and tracer are all kept in resident memory and do not have to be loaded when required.

PISTOL was written, it seems, in C, thus allowing the kernel to be increased to include as many primitives as required for increased speed or versatility. Naturally, this would require quite a bit of knowledge, not only of machine language and operating system uses, but also of the PISTOL ideology and implementation.

PISTOL is available in CP/M 80 from SIG/M Volume 114 (Version 2.0). When compared to Forth, it is difficult to say which would be the eventual favorite. I use both, and really treat them as two separate, although related, languages. They tend to complement each other very well. Check it out, and see whether it is of any interest to you.

That, unfortunately, is all the space I have. I want to remind you to check out the *COMPUTER LANGUAGE* Bulletin Board and the new section on CompuServe, SYSOPed by your humble narrator. If you want to leave a message on CompuServe, my user ID is 76703,762. Both should be chocked full of goodies by the time you read this! (Otherwise I'm out of a job.) Till we meet again . . .

Useful addresses: SIG/M is at P.O. Box 2085, Clifton, N.J. 07015-2085. CP/MUG is at 1651 Third Ave., New York, N.Y. 10028. PC-SIG is at 1556 Halford Ave., Suite 130, Santa Clara, Calif. 95051, (408) 730-9291. ■

EXOTIC LANGUAGE OF THE MONTH CLUB

Occam: A parallel processing language from the U.K.

By Anthony Skjellum

The idea of parallel computation is not new to the computer field. Large computers such as the Cray-1 often incorporate a pipelined architecture to improve performance for vector-oriented operations. Other computers offer floating point accelerators, which operate along similar lines.

Both of these concepts revolve around a central processor with a parallel-processing attachment. An alternate approach would be to eliminate the single, fast, expensive central processor and design a computer with many small, slower processors.

With advances in microprocessor and VLSI (very large scale integration) design, it is becoming feasible to create such computers. However, three major problems exist. The first is the need for convenient processors to use in building the parallel processor. The second problem is the need for a convenient medium-level language for managing the resources offered by such a computer. Finally, only programs that take advantage of the parallel design can hope for enhanced performance compared to a traditional algorithm run on a serial computer.

Inmos Ltd., a U.K.-based company, is dedicated to creating parallel-computer chips. The company's current version is known as the IMS T424 transputer. These chips contain a microprocessor, random-access memory, external memory interface, and high-speed inter-processor communication channels on a single die.

While the transputers are not yet in production, they will soon be available. When available, design of parallel computers will be greatly simplified. Furthermore, Inmos has arrived at a means for solving the software requirement mentioned previously. It has done so by defining and creating a language called Occam.™

Occam is a medium-level language that incorporates inter-processor communication and parallel processing as part of its structure. Its inherent generality could make it the de facto standard. This article explores Occam in detail and includes parallel programming examples. Information about Inmos, Occam evaluation products, and related matters is included at the end of this article.

Imagine the following problem: multiply *N* pairs of numbers all of which are known initially. This is an inherently parallel task. We consider a very traditional approach to the problem at the following hypothetical company.

If we had *N* clerks multiply one pair each, the whole job would be completed in the time required for the slowest clerk to do the arithmetic. In Occam notation, this operation would be requested as follows:

```
PAR i = [0 FOR N]
  a[i] := b[i] * c[i]
```

Assuming that clerks demand high wages for this mindless multiplication task, we find it more economical to use a serial computer instead of clerks. However, we notice immediately that the computer program we must write incorporates serial evaluation of the multiplications.

Despite our initial reservations, we use the computer because it handles the whole serial task faster than our fastest clerk can perform a single multiplication. Time is saved as well as money. The serial operations could be presented in Occam as follows:

```
SEQ i = [0 FOR N]
  a[i] := b[i] * c[i]
```

Eventually our hypothetical company grows and the quantity *N* grows too. Soon the original computer system proves too slow for the task of multiplication. Management concedes this point and buys a second computer. Now the task is divided equally between the two systems: they have discovered rudimentary parallel processing. The Occam representation of the new procedure is:

```
PAR                    —(assume N is even)
  SEQ i = [0 FOR N/2]
    a[i] := b[i] * c[i]
  SEQ j = [N/2 FOR N]
    a[j] := b[j] * c[j]
```

Clearly, this procedure uses half the time required by the single computer.

Alternatively, a single larger computer could have been purchased. However, when *N* gets very large, no computer will be fast enough to do the job in the time required by our rather fussy managers. Realizing this limit, the company buys a parallel computer that can actually handle the operations:

```
PAR i = [0 FOR N]
  a[i] := b[i] * c[i]
```

in parallel. Now, the only way to gain additional speed is for the single processors involved to work faster. No additional benefit can be gained by dividing the problem between two parallel computers as depicted by the following Occam blurb:

```
PAR           -- (assume N is even)
  PAR i = [0 FOR N/2]
    a[i] := b[i] * c[i]
  PAR j = [N/2 FOR N]
    a[j] := b[j] * c[j]
```

We have used the multiplication example to introduce the idea of parallel processing and also to exhibit some of the elements of Occam. The *SEQ* and *PAR* keywords are called constructs, while sequences like

```
var = [low FOR high]
```

are called replicators.

It should be noted that Occam uses indentation as part of the grammar, with indentation quantized to two-space increments. (Continuation lines are also permitted and must be indented more deeply than the initial line.) Furthermore, any operation such as an assignment or procedure call is known as a process. Finally, Occam keywords must always be capitalized. Conversely, variables and constant names may be of mixed case and include numbers and periods, except for the first character, which must always be alphabetic.

Occam allows procedures similar to traditional languages. It does not support functions or recursion as part of the fundamental language. Defining an actual procedure will illustrate many additional Occam features. Such a procedure is presented in Listing 1.

Let's review the many features used in the listing. First, we defined the procedure:

```
PROC poly(VAR x,VALUE retn) =
```

Arguments to a procedure are either passed by value (VALUE) or address (VAR). The latter variety allows for transmission of information to the calling routine. Arguments denoted as VALUEs cannot appear on the left-hand side of an assignment.

Next, we have a single constant definition, followed by two variable declarations:

```
DEF LIM = 5:
VAR i:
VAR temp:
```

In the current incarnation of Occam (known as proto Occam), the only variable types are (32-bit twos complement) integers, arrays of integers, byte arrays, channels and arrays of channels. Channel variables permit inter-processor communication and are discussed fully later in this article. (A new Occam version that includes general data types is anticipated.)

Let's digress a moment to discuss scope rules. As might be expected, the scope of variables is restricted to the "block" of code in which they are defined. Thus the *i* and *temp* variables are defined throughout the *poly()* procedure, while *num* and *fact* are defined only within the *WHILE* loop.

After the variable declarations, a *SEQ* construct is present. As before, this indicates a series of processes that are to be performed sequentially. The next two lines are assignment processes:

```
i    := 0
temp := 0
```

They perform the obvious operations of assigning zero to *i* and *temp*. These lines are noteworthy for two reasons. First, ":=" is used in Occam for assignment and "=" is used in logical expressions. Second, we cannot assume any automatic initialization (e.g., zeroing) of variables will be performed by Occam.

After the initialization steps, the procedure invokes a *WHILE* construct. The syntax of the construct is as follows

```
WHILE <expr>
    <process>
```

where <expr> is a valid Occam expression, and <process> is either a single process line or a construct followed by a series of processes.

Expressions deserve some independent comments. First, there is no hierarchy in Occam expressions. Grouping with parentheses is necessary for an expression like

```
a + b * c
```

to be legal. Thus, it must be written as either

```
a + (b * c)
```

or

```
(a + b) * c
```

to be accepted at compile time. In addition to the standard arithmetic operators, Occam provides the logical operators /\ (or), \/ (and), and >< (exlusive or) as well as modulus (\ ). Logical NOT, the boolean OR and AND operators are included as are shifting operators (<<), (>>).

Before abandoning the example in Listing 1, three more items need to be covered. The first is the *IF* construct. Occam makes efficient use of *IF* by making it a combined conditional and case statement. *IF* is followed by a set of logical expressions, only one of which is executed. If none are true, the process is stopped. Therefore, *IF*s normally include a *TRUE* condition that acts like the default clause of a case statement. Formally, the *IF* statement has the following form:

```
IF
    <expr1>
        <process1>
    ...
    <exprN>
        <processN>
```

The final items are comments and procedure termination. As you've probably already noticed, Occam comments are initiated with two dashes (--). Second, all procedures must end. In Occam they are terminated by a colon. This colon appears at the end of the last process in the procedure. In our example, the colon appears at the end of the assignment process

```
n := temp:    -- store return value
```

Since procedures can be defined within other procedures, the terminating colon turns out to be necessary.

```
-- compute sum of (x^n)/n! and return in retn
PROC poly(VALUE x,VAR retn) =
  DEF LIM = 5:          -- terms in polynomial (a constant)
  VAR i:                -- looping variable
  VAR temp:             -- holds partial sum.
  SEQ
    i    := 0
    temp := 0
    WHILE i < LIM
      VAR num:          -- temporary for numerators
      VAR fact:         -- factorials temporary
      SEQ
        IF
          (i = 0)
            SEQ
              fact := 1
              num  := 1
          TRUE
            SEQ
              fact := fact * i
              num  := num  * x
        temp := temp + (num/fact) -- add in next factor
        i    := i + 1             -- increment loop
    retn := temp:                 -- store return value
```

Listing 1.

When we discussed the hypothetical company, we used Occam blurbs to describe parallel operations. Then we turned to Listing 1, which was completely serial. Now that we've introduced some of the traditional aspects of the Occam language, we can explore parallel operations and communication.

Communication is the key to parallel processing, and Occam supports interprocessor dialogs through channels. Channels are one-way streams that send synchronized information from one processor to another. For example, the following procedure squares the data read from the channel "input" and transmits it to the channel "output".

```
PROC square(CHAN input,output) =
  VAR x,y:
    SEQ
      input ? x       -- read x
      y := x*x        -- compute y
      output ! y:     -- write output
```

About channels: they are physical or logical connections between processes, and only two processes reference a specific channel—one reads from it and the other writes to it. The operators ? and ! perform channel reading and writing respectively. Colloquially they are often referred to as grab and bang, but these are not there official names. Note that the unit of channel transmission is the byte, however, current Occam versions work strictly with 32-bit data chunks.

Now let's talk about parallel operations. We want to consider a parallel version of the algorithm presented in Listing 1. The parallel version is presented in Listing 2.

You may find Listing 2 to be somewhat of a shock. Initially I couldn't sort out anything when I looked at an Occam routine. Let's examine the whole procedure carefully so that the important points become clearer.

The procedure *poly0* is defined within *poly()*. Thus, the first code we must consider occurs at the end of *poly()* (Listing 3).

Figure 1 shows an illustration of the processes involved and the channels they own.

Within *poly0*, some concepts need examination. First, the operation

```
PAR i = [0 FOR LIM]
    <process>
```

defines a set of independent processes, each with its own local variables. The *i* variable is unique for each of the processors (0, 1, 2, 3, or 4 in our case). Occam doesn't require us to define variables used in replicators: in the *poly0* routine, we never define *i*.

Second, how does communication degrade performance? Each processor must wait for its predecessor before it can compute its term of the sum. Therefore, the whole parallel operation takes as long as a single processor working by itself.

However, five partial problems can be in progress at any time, so the pipeline of processors presented in Listing 2 handles one calculation only as fast as a single processor, but it also handles five calculations as fast as a single processor. Thus, we have found a way to gain speed but only through careful use of the pipeline.

Earlier we looked at parallel multiplication. There was no communication between processors, so an *N*-pair multiplication completed in the time needed for a single multiplication. This arrangement offers *N*-fold improvement for a single calculation. This is conceptually distinct from the type of improvement offered by the *poly()* pipeline arrangement.

```
-- compute polynomial in parallel
PROC poly(VALUE x,VAR retn) =
  DEF LIM = 5:
  CHAN common[LIM+1]:             -- means for transmitting data
  -- workhorse routine
  PROC poly0 =
    SEQ
      PAR i = [0 FOR LIM]
        VAR xx:                   -- copy of x
        VAR temp:                 -- partial sum
        VAR num:                  -- numerator  temporary
        VAR fact:                 -- factorials temporary
        SEQ
          common[i] ? xx;temp;num;fact  -- read last values
          IF
            (i <> 0)
              SEQ
                fact := fact * i
                num  := num  * xx
            TRUE
              SEQ
                fact := 1
                num  := 1
          temp := temp + (num/fact)       -- new temporary
          common[i+1] ! xx;temp;num;fact: -- write new values
  PAR
    poly0                           -- perform the operations
    common[0]    ! x;0;0;0          -- feed initial input
    common[LIM] ? x;retn;ANY;ANY: -- extract output
```

Listing 2.

```
[1]          PAR
[2]            poly0                           -- perform the operations
[3]            common[0]    ! x;0;0;0          -- feed initial input
[4]            common[LIM] ? x;retn;ANY;ANY: -- extract output
```

Line [1] causes the three subsequent processes to be executed in parallel. Line [2] activates poly0, while line [3] provides initial data to the zeroth channel of common[]. This data will be picked up within poly0. Line [4] extracts the result when it filters into common[LIM]. Since the inputs and outputs are synchronized, the last line waits until the last process of poly0 is complete. Note that the ANY variable is defined by Occam as a sink for information.

Listing 3.

```
(Assume LIM = 5)

Trivial Write:
Trivial Read:
-----------------      ----------          ----------
-----------------
|common[0] ! ...| ---> |poly0 #0| ... ---> |poly0 #4| --->
|common[5] ? ...|
|               |      |        |          |        |      |

| Owns:         |      |Read  0 |          |Read  4 |      |
Read  5 |
|      Write 0  | ---> |Write 1 | ... ---> |Write 5 | ---> |
|
-----------------      ----------          ----------
-----------------

Notes: Read/Write numbers indicate subscript of common[] array.
The indication poly0 #3 indicates that i = 3 for this process,
etc.
```

Figure 1.

```
VAR copy.number:
SEQ
  copy.number := 1     -- initially 1 copy
  WHILE TRUE
    SEQ
      copyno ! copy.number  -- tell copier
      ALT
        plus ? ANY         -- using ANY means we dump the value
          SEQ
            copy.number := copy.number + 1
        minus ? ANY
          SEQ
            copy.number := copy.number - 1
        reset ? ANY
          SEQ
            copy.number := 1
```

Listing 4A.

```
VAR copy.number:
SEQ
  copy.number := 1     -- initially 1 copy
  WHILE TRUE
    SEQ
      copyno ! copy.number  -- tell copier
      ALT
        (copy.number < 20) & plus ? ANY
          SEQ
            copy.number := copy.number + 1
        (copy.number > 0)  & minus ? ANY
          SEQ
            copy.number := copy.number - 1
        reset ? ANY
          SEQ
            copy.number := 1
```

Listing 4B.

Sometimes a process must wait for the first input from one of several channels. The *ALT* (alternative) construct offers a means for this selection.

Imagine that we have a photocopier with three buttons. The first is "plus," which means add one to the copier count. Analogously, a "minus" button exists and subtracts one from the copier count. The third button is "reset," and resets the count to one. The copier isn't interested in making zero copies and cannot make more than 19 at a time.

Our goal is to design an Occam process that controls inputs from the three buttons and modifies a variable called *copy.number* accordingly. We must also output any change in *copy.number* to the output channel "copyno" so that the copy machine itself will be properly set. To begin, let's ignore operating limits. The simple-minded program that works without limit checks is presented in Listing 4A.

Ignoring the operating limits is not truly a satisfactory arrangement. To incorporate these limits, we use Occam Guards. Guards are conditions that must be satisfied in order for an *ALT* condition to be executed. A Guard has the form:

condition & channel ? variable-list

where *condition* is a logical expression. Making use of this new feature, we design a new blurb in Listing 4B.

Next we want to be more fancy. We want to set up a priority to the alternative selection process. If the reset button is pushed at the same time as plus or minus, we want the reset to occur first. Occam supports this through prioritized alternatives. Listing 4C is a prioritized version of Listing 4B.

In Listing 4C, reset has the highest priority and minus has the lowest. We would actually prefer plus and minus to have equal priority. This change is reflected in Listing 4D. Evidently, the guarded sequence (TRUE) & SKIP (or just SKIP) always executes whenever selected.

In addition to *PRI*oritized *ALT* constructs, Occam supports prioritized *PAR* constructs. This allows transputers and other processors to give different priorities to time-shared processes that run on the same processor unit. For example, a background process might exist on every processor. It would have to have highest priority in order to provide its services most effectively. Such an arrangement could be handled as follows:

```
PRI PAR
  background
  PAR
    foreground1
      ...
    foregroundN
```

where *background* has the highest priority and the others have equal (but lower) priorities than *background*. One use for a background processor would be high-level data packet transfer between arbitrary processors in a system.

A special channel called TIME is available to all processes. When read, it provides a processor with a value derived from a freely running clock. A process can wait for a specified time interval using a sequence like:

```
VAR now:
  SEQ
    TIME ? now
    TIME ? AFTER now+10
```

which waits for 10 clock units. The time-out sequence may be used in a Guard to an *ALT* process. In this connection, convenient time-out capability is afforded.

In a real system, physical control over channels and processes must be possible. Occam defines the *PLACED PAR* construct to permit specific location of processes on physical processors. The information needed to complete a physical specification depends on the system used, so I won't elaborate further on *PLACED PAR* and its parameters.

Occam is a medium-level language that provides powerful parallel programming constructs. Its philosophy is dictated by Occam's Razor: the simplest complete solution to a problem is the correct one.

Occam does not incorporate myriad features. It concentrates on the basics and is successful. Its one great flaw is a lack of data types such as real numbers, but I expect that Inmos will upgrade the language to overcome this intentional omission. Expect Inmos to concentrate on silicon; Occam is merely Inmos's means to bridge the hardware-software gap.

For those users who want to learn about parallel processing, the OPS system offers a professional means to begin programming in Occam. It is actually possible to produce usable transputer software with the OPS. When they become available, transputer systems may very well

```
VAR copy.number:
SEQ
  copy.number := 1      -- initially 1 copy
  WHILE TRUE
    SEQ
      copyno ! copy.number  -- tell copier
      PRI ALT
        reset ? ANY    -- handle as priority
          SEQ
            copy.number := 1
        (copy.number < 20) & plus ? ANY
          SEQ
            copy.number := copy.number + 1
        (copy.number > 0)  & minus ? ANY
          SEQ
            copy.number := copy.number - 1
```

Listing 4C.

NEW RELEASE

Eco-C Compiler
Release 3.0

We think Rel. 3.0 of the Eco-C Compiler is the fastest full C available for the Z80 environment. Consider the evidence:

Benchmarks*
(Seconds)

Benchmark	*Eco-C*	Aztec	Q/C
Seive	29	33	40
Fib	75	125	99
Deref	19	CNC	31
Matmult	42	115	N/A

**Times courtesy of Dr. David Clark*
CNC - Could Not Compile
N/A - Does not support floating point

We've also expanded the library (120 functions), the user's manual and compile-time switches (including multiple non-fatal error messages). The price is still $250.00 and includes Microsoft's MACRO 80. As an option, we will supply Eco-C with the SLR Systems assembler - linker - librarian for $295.00 (up to six times faster than MACRO 80).

For additional information, call or write:

ECOSOFT INC. *(317) 255-6476*
6413 N. College Ave. • Indianapolis, Indiana 46220

VISA MasterCard

CIRCLE 22 ON READER SERVICE CARD

```
VAR copy.number:
SEQ
  copy.number := 1     -- initially 1 copy
  WHILE TRUE
    SEQ
      copyno ! copy.number  -- tell copier
      PRI ALT
        reset ? ANY   -- handle as priority
          SEQ
            copy.number := 1
        (TRUE) & SKIP
          ALT
            (copy.number < 20) & plus ? ANY
              SEQ
                copy.number := copy.number + 1
            (copy.number > 0)  & minus ? ANY
              SEQ
                copy.number := copy.number - 1
```

Listing 4D.

already have a large body of software available for them because of OPS.

While OPS really screams for a support library, the job of writing support routines is in itself an instructive introduction to the language. I have not yet acquired any of the user group offerings, but I expect that they will be very helpful to anyone planning to use Occam.

I've discussed most of Occam's features, but some items have received only a small degree of attention or none at all. For example, Occam has a convenient way to handle constant tables, and it handles strings of characters in a reasonable way. I chose to concentrate on the more important aspects of Occam: its ability to provide a parallel programming environment.

I'd also like to make a few comments about Inmos and provide some related information.

Inmos is a company owned by the British government. I first heard about it through a short article in the May 12 issue of *Fortune*. The British Consulate in San Francisco, Calif., provided the company's U.S. and U.K. addresses. They are: Inmos Inc., P.O. Box 16000, Colorado Springs, Colo. 80935, U.S.A., tel. (303) 630-4000, telex 910 920 4904 and Inmos Ltd., Whitefriars, Lewins Mead, Bristol BS1 2NP, U.K., tel. (0272) 290861, telex 444723.

I had the opportunity to discuss Occam with Colin Whitby-Strevens from the Bristol office of Inmos, who visited me at the California Institute of Technology (Caltech) early in September. He answered many of my questions and was very helpful. I'd like to thank him for his assistance and input.

Whitby-Strevens informed me that an Occam users group exists and is supported by Inmos. The secretary for the group is Michael Poole, who may be reached at the Bristol address. He can be contacted concerning the group's offerings. I understand that a floating point package is one of the available items.

I look forward to hearing from other Occam users. ■

Anthony Skjellum graduated from Caltech with a B.S. in physics. He is now pursuing graduate studies in chemical engineering, exploring the use of parallel computers and distributed processing for chemical engineering applications.

THE PROGRAMMER'S SHOP™

helps compare evaluate and find products. Get answers.

SERVICE: FINDING PRODUCTS

Don't know of a product or want a better one? If the need is for developing micro software, we usually can suggest or find alternatives.

"C" Language

	LIST PRICE	OUR PRICE
MSDOS: C86 - 8087, reliable	$395	call
Desmet by CWare with debugger	159	145
Lattice 2.1 - improved - 30 addons	500	call
Microsoft C2.x	500	349
Williams - NEW, debugger	500	call
Instant C Interpreter, fast, full	NA	500
CPM80: Ecosoft C-now solid, full	250	225

UNIX PC

	Runson		
COHERENT-for ''C'' users	PClike	$500	475
COHERENT-NCI-tailored, fast	MSDOS	NA	695
VENIX-''true V7'' w/FTN	PClike	800	775
XENIX-''true S3''-rich, C-MSDOS	PC	1350	1285

EDITORS

BRIEF - Intuitive, flexible	PCDOS	NA	195
PMATE - powerful	8086	225	195
VEDIT - full, liked	8086	150	119

Feature

Prolog-86-Learn quickly, experiment with this AI language Examples. MSDOS. $125.

Recent Discovery

SCIL-Manage versions, change to source code, documentation. Minimize confusion, disk space. Interactive. CPM 80, MSDOS $349.

FORTRAN

	Runson	LIST PRICE	OUR PRICE
MS Fortran - Improved	MSDOS	350	255
Intel Fortran - 86	IBMPC	NA	1400
DR Fortran-86 - full '77'	8086	500	349
PolyFORTRAN - XREF, Xtract	PCDOS	NA	165

SUPPORT PRODUCTS

LANGUAGES: IQ LISP	PCDOS	175	call
MicroProlog	MSDOS	NA	285
HS/FORTH - fast	MSDOS	220	210
LIBRARIES: BTRIEVE ISAM	PCDOS	245	215
Greenleaf C - thorough	MSDOS	NA	165
HALO Graphics - fast, full	PCDOS	200	175
TOOLS: Disk Mechanic-rebuild	MSDOS	70	65
MULTILINK - multitask	PCDOS	295	265
Polylibrarian-thorough	MSDOS	99	89
PolyMAKE-compiles	PCDOS	99	89
Profiler-86-easy to setup	MSDOS	NA	125
XShell-add IF-THEN-ELSE	MSDOS	225	215

Call for a catalog and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™
128-L Rockland Street, Hanover, MA 02339
Visa Mass: 800-442-8070 or 617-826-7531 MasterCard

Note: All prices subject to change without notice.

Mention this ad. Some prices are specials.

All formats available.
Ask about POs, COD.

CIRCLE 17 ON READER SERVICE CARD

COMPUTERVISIONS

A chat with Gary Kildall, founder of CP/M

By Regina Starr Ridley

Gary Kildall, inventor of CP/M and founder and chairman of Digital Research Inc., is a soft-spoken, unpretentious man.

His office is rather ordinary and looks like it belongs to someone who works hard. Computer hardware is strewn about and numbers are scrawled on a blackboard on one wall.

Large windows face a hill covered with pine trees—unlike many of his peers, Kildall chose to build DRI's headquarters in Monterey, one of California's most beautiful coastal towns and a few hours drive from less aesthetically appealing Silicon Valley.

Kildall is one of the few individuals to whom the cliche "he revolutionized the microcomputer industry" actually applies. But looking at the 42-year-old Kildall—trim, tanned, freckled, and dressed in casual clothes—it's easier to imagine him as an outdoorsman than a computer scientist.

But he is indeed quite a computer scientist. He also is a major force behind the direction Digital Research is taking in the current, tempestuous market. The course he is steering reflects one man's insights into what the future in software and operating systems may hold.

Originally Kildall's primary interest was in computer languages, and his goal was to teach computer science. He received a B.S. in numerical analysis and an M.S. and Ph.D. in computer science from the Univ. of Washington. His thesis work was on compiler code optimization—theoretical approaches to doing optimal code generation. While at school he did maintenance work on Burroughs' ALGOL compiler.

After receiving his Ph.D. in 1972, Kildall went on to teach general computer science and compiler courses at the Naval Postgraduate School in Monterey. During the one day a week and one quarter a year allotted for consulting work, Kildall worked for Intel.

"During that time at Intel I became interested in microcomputers," said Kildall. "Micros were just starting out and they didn't have any software at all—just really very basic software. I started to do some universal software tools for Intel, simulation of their machines on the bigger computer and various things that were needed to get things off the ground."

Kildall became interested in trying to get a high-level language around microcomputer software development, not for end users but for people who were trying to write software.

That interest prompted the development of PL/M (Programming Language for Micros), a derivative of a XP/L, a compiler writing language. His work on PL/M led directly to the development CP/M (Control Program for Micros).

"CP/M was actually a follow-on product in support of PL/M and not intended to be a product in its own right," Kildall chuckled. "We needed to have some kind of an operating system to be the foundation for running a program that would support PL/M. That's why the name is the same basically."

"At that time I didn't think much about CP/M other than it was nice people were interested in it," said Kildall. "The people who I was working with sort of realized that there was something there but that it was going to take some time before it actually caught on."

Kildall worked for about a year trying to get PL/M on a microcomputer, one of the Intel development systems. "It was going along pretty well," said Kildall, "but Intel was going off in its own direction." Intel had started its own PL/M compiler development internally, separate from Kildall's work. Intel had also started its own operating system development called Isis, which, said Kildall, was very much like CP/M.

"Intel decided that its big philosophy in selling software was that it would use that software to sell its little blue boxes. Intel said, 'we don't want to unbundle our software because that way somebody else could come along with a look-alike blue box and then undercut us in price,'" said Kildall.

"Intel was selling those development systems for $25,000 a piece and doing really well with them. So if it had sold Isis, which was essentially equivalent to CP/M, with what originally was a $60 to $70 price tag, it could have completely wiped out the blue box sales."

DRI was founded in 1976 because of interest in CP/M. "Some people had used it and, you know, people liked it." Kildall said modestly. "There were a lot of different operating systems with different tangential kinds of features, and they all had a variety of faults. CP/M just happened to be simple but useful." It took about nine months to put together CP/M itself and about a year to complete the programs that had to go with it. Kildall officially left the Postgraduate School in 1978.

About that time, the PL/I standardization committee was meeting in Carmel, Calif., and had just finished the first PL/I ANSI standardization specification. The committee was trying to produce two subsets, subset G for general purpose and another subset for real time.

"PL/M was a dialect of XP/L, and XP/L had been a dialect of PL/I, and they were all intermixed as far as the syntax and their general appearance," Kildall said. "But there was very little else that was similar besides the common family tree. I did realize at that point that the people working on the PL/I subset G were doing a really nice thing. They knew that the full language was not going to be supported on a small computer—it just wouldn't fit, that was all."

Kildall began to feel that PL/I subset G

NGS FORTH

A FAST FORTH OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MSDOS COMPATIBLES.

- *79 STANDARD
- *FIG LOOKALIKE MODE
- *PC-DOS COMPATIBLE
- *ON-LINE CONFIGURABLE
- *ENVIRONMENT SAVE & LOAD
- *MULTI-SEGMENTED
- *EXTENDED ADDRESSING
- *AUTO LOAD SCREEN BOOT
- *LINE AND SCREEN EDITORS
- *DECOMPILER & DEBUGGING AIDS
- *8088 ASSEMBLER
- *BASIC GRAPHICS & SOUND
- *NGS ENHANCEMENTS
- *DETAILED MANUAL
- *INEXPENSIVE UPGRADES
- *NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICE: $70

PLEASE INCLUDE $2 POSTAGE & HANDLING WITH EACH ORDER.
CALIFORNIA RESIDENTS : INCLUDE 6.5% SALES TAX.

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

CIRCLE 45 ON READER SERVICE CARD

was a reasonable language for doing work with small computers. And CP/M needed to have an applications language of some sort. So he decided to do an implementation of PL/I subset G. That's where the Digital Research PL/I compiler started off.

Kildall thought the compiler wouldn't be too difficult to do and would take about nine months. But, he said ruefully, it wound up taking two years.

The project also didn't have the impact he thought it would on the applications users. "The difficulty we had was that the machines we were working with still had relatively small memory systems. And the difference between high-level code vs. assembly language code was still significant to people," Kildall said.

"I think it was a good project and definitely worthwhile, but I sort of felt that the 8086 chip was going to be more popular sooner than it was," he added. "It was out in about 1978 and showed real promise, but it just didn't get picked up. And we really couldn't sell the idea of high-level language coding on small computers until we had a bigger memory system. So now that the 8086 and that whole family is popular, there's not such an emphasis on compactness of approach and people are writing applications using high-level languages now."

Since coming out with CP/M in 1978, DRI has introduced CP/M 86, CP/M for a 16-bit computer; MP/M 86, multi-user CP/M for a 16-bit computer; Concurrent CP/M, which has multitasking, networking, real-time, and windowing capabilities; and Concurrent PC-DOS, which has the same capabilities as Concurrent CP/M but can be used on IBM micro systems.

MS-DOS and PC-DOS hit CP/M 86 pretty hard in the marketplace. The problem was, according to Kildall, CP/M 86 was not meant to be DRI's primary product—it was to be used as a stepping stone to MP/M and Concurrent CP/M.

"I misjudged the timing somewhat—it wasn't really until this year that people started thinking multitasking was important," said Kildall. "Now IBM has announced Top View with multitasking, emphasizing that multitasking is an important concept. So most people say 'Oh, IBM says that multitasking is an important concept, so it must be an important concept,' " Kildall laughed.

Kildall didn't feel that the high price of CP/M 86 compared with MS-DOS was an important issue. "If we had been prepared with a CP/M 86 strategy we would have been able to do the pricing right, and we would have been prepared for the IBM phenomenon. We were prepared for a multitasking phenomenon."

"We figured," he said, "we have a megabyte of memory, what are we going to do with that megabyte of memory? Most applications were not going to use a million bytes of memory. We figured that people wanted an operating system with more functionality than you had in 8-bit computer operating systems."

"So now we've reached a situation where IBM has endorsed multitasking and multi-users. But they're saying, 'we're going to give you these things, but we don't have the tools right now.' The best thing for them would be to have a PC-DOS that had multitasking and multi-user capabilities. And if they had that they wouldn't announce anything else. Top View gives you multitasking, real time, networking, and multi-using."

"The thing is that's exactly the product that we were building," said Kildall. "IBM's endorsement has caused a lot of OEMs to come back to us and say, 'I can now endorse your product because I've realized the importance of Concurrent CP/M.' "

"This is the basis for the interest in Concurrent CP/M and Concurrent DOS. And we have other products beyond this that haven't been announced. Then there are follow-on products that are just kind of progressive steps, adding facilities. I want to be careful of what I say, because I don't want to say anything about something that hasn't been released yet."

According to Kildall, Concurrent CP/M and Concurrent DOS are doing very well at the OEM level, which he feels is the most important indicator. He considers earnings from Concurrent to be a very significant part of DRI's revenue.

"We really believe in Concurrent, there's no question about that," said Kildall. "But it does come back entirely to the fact that you can be successful only if you have the backing of the large manufacturers nowadays. And that's exactly what we're working on doing."

Concurrent is actually the sixth generation of the CP/M operating system, Kildall pointed out. "When you take a look at some things like Top View and UNIX and so forth, you see they're sort of half-baked in a micro sense because they can't really perform those low-level functions as effectively as Concurrent."

Kildall breaks down UNIX into three important elements: the operating system, the C language, and the standard run-time library. He feels the major contribution UNIX has made has been in the standardization of the run-time library and the C language.

"If you're careful and write C source code in a machine-independent way, and you set up your library so that it matches the UNIX standard run-time subroutines, then you can get transportation from one processor or one operating system to many different processors or many different operating systems without any major recoding," he said.

Kildall is not nearly so supportive of the UNIX operating system, which he said is pretty well known in the industry as being "loosey-goosey"—not a very tight system. He has also experienced many problems with it in terms of reliability and clarity of the user interface.

"You talk to anybody who's a UNIX user, and they will say 'UNIX is great but it has a lot of problems in terms of commercialization of the operating system.' You can go to the C language and say, yes, there's lots of things I would have done differently. But the fact of the matter is that in spite of its shortcomings, the C language and the run-time library give you transportation, and that's something which is very, very valuable."

"There are lots and lots of operating systems around. The UNIX operating system itself doesn't have any inherent new technology. And that's why I'm not as hot on the operating system as I am on the language and the standardization of the run-time system."

"We certainly are working with the UNIX phenomenon in the sense that we are offering portable software. And this goes back to my original comment about C and the standard run-time library. Our software now will run on virtually any operating system we choose, and one of the targets is UNIX. Now, as UNIX becomes popular with various people, we can offer our software products on UNIX. Now that's where the money is."

"The intention is to make all of our products portable through C and its run-time library, which of course makes them immediately portable to UNIX because that's C's original home. Our initial support of UNIX is through what we call the UNIX application library that we've constructed for AT&T."

Transportation of a high-level language like PL/I or the C compiler itself is a much more difficult task than porting over applications like DR Logo, for example, said Kildall. DR Logo transports very, very rapidly because there are few machine dependencies.

DR Logo is DRI's main product on the education market. Kildall saw Logo come around and was very interested in it because of the way the language used LISP concepts—recursion, list processing and homogeneity.

"I personally never really liked BASIC at all," said Kildall. "It was a language that came from FORTRAN, the early days of FORTRAN. It was not intended to be a general language used by the masses."

Kildall has always felt that BASIC was not a good learning model because the use of numbers is a very limited concept that in turn enforces a limited style of programming. "It's very difficult to make leaps from that into something with more general concepts."

Kildall prefers using LISP to teach programming because it has the most general concept of what programming should be because it's all symbolic. "You can literally do almost any kind of operation that you want very quickly and easily," he said.

The problem with LISP, said Kildall, is its very user unfriendly syntax. He didn't want to bring LISP to microcomputers because its unfriendly nature would turn people off to using it.

Logo was originally intended to get rid of some of the unfriendly nature of the front end and still get all the power of LISP. Graphics were added, Kildall said, because you could use them for a kind of a bait and switch. First people would be tempted with the graphics, and then they could discover all the other things that could be done with LISP.

The push behind DR Logo was "let's get in there and try to do a personal computer Logo and get people away from the concepts of BASIC as much as we can. Let's get rid of BASIC and go with a language that actually gives people a tool to think with, rather than make a person first learn concepts that have little to do with the problem they're solving," Kildall said.

Support of Logo has been slowly building, said Kildall. "It takes a year or two before you can really tell if something is taking off or not. DR Logo has moved very nicely and sales through OEMs are picking up."

DRI has pulled back somewhat from the education market. "We have closed up a lot of things at the retail level. For example, with DR Logo we're not offering anything more than our original version on the retail level basically because price cutting at the consumer level has been so dramatic that we can't make any money on it. We decided to put all our efforts into OEMs."

"We tried to get the consumer business going. What we're getting is that every time we came up with a product like DR Logo in the consumer division, our OEM interest in it was so much greater that retail was of little interest. We take three-quarters of our people working on retail and one-quarter on the OEM level, and the one-quarter makes 10 times as much as the three-quarters. So we just decided to slack off a bit."

Kildall sees the principle market for DRI as being the high-end commercial market. "It's unfortunate but it goes back

Pascal and C Programmers

Your programs can now compile the *FirsTime*™

FirsTime is an intelligent editor that knows the rules of the language being programmed. It checks your statements as you enter them, and if it spots a mistake, it identifies it. *FirsTime* then positions the cursor over the error so you can correct it easily. *FirsTime will identify all syntax errors, undefined variables, and even statements with mismatched variable types.* In fact, any program developed with the *FirsTime* editor will compile on the first try.

Unprecedented
FirsTime has many unique features found in no other editor. These powerful capabilities include a zoom command that allows you to examine the structure of your program, automatic program formatting, and block transforms.

If you wish, you can work even faster by automatically generating program structures with a single key-stroke. This feature is especially useful to those learning a new language, or to those who often switch between different languages.

Other Features: Full screen editing, horizontal scrolling, function key menus, help screens, inserts, deletes, appends, searches, and global replacing.

Programmers enjoy using *FirsTime*. It allows them to concentrate on program logic without having to worry about coding details. Debugging is reduced dramatically, and deadlines are more easily met.

FirsTime for PASCAL	$245
FirsTime for C	$295
Microsoft PASCAL Compiler	$245
Microsoft C Compiler	$395
Demonstration disk	$25

Get an extra **$100 off** the compiler when it is purchased with ***FirsTime.***
(N.J. residents please add 6% sales tax.)

Spruce
Technology Corporation
110 Whispering Pines Drive
Lincroft, N.J. 07738
(201) 741-8188 or (201) 663-0063

Dealer enquiries welcome. Custom versions for computer manufacturers and language developers are available.

FirstTime is a trademark of Spruce Technology Corporation.

CIRCLE 33 ON READER SERVICE CARD

to when you try to keep a business going and the profit level at a reasonable rate, the products that are bringing in the most revenue are the ones people will gravitate toward. The difficulty in concentrating in educational products—which I'd like to—is that there is a much larger margin in commercial software."

Kildall seems to have found an equilibrium between the computer scientist, the businessman, and the human being. Having leadership with this kind of balanced outlook makes DRI's future look promising. ■

Regina Starr Ridley is the managing editor of COMPUTER LANGUAGE.

Thunder Software

- **The THUNDER C Compiler** - Operates under the APPLE Pascal 1.1 operating system. Create fast native 6502 programs to run as stand alone programs or as subroutines to Pascal programs. A major subset of the C defined by K & R. Includes a 24 page users guide, newsletters. Macro preprocessor, runs on APPLE][,][+, //e, //c. Source code for libraries is included. **Only $49.95**
- **ASSYST: The Assembler System** - A complete 6502 editor/assembler and lister for APPLE DOS3.3. Menu driven, excellent error trapping, 24 p. users guide, demo programs, source code for all programs! Great for beginners. **Only $23.50**
- **THUNDER XREF** - A cross reference utility for APPLE Pascal 1.1. XREF generates cross references for each procedure. Source code and documentation provided. **Only $19.95**

Thunder Software POB 31501 Houston Tx 77231 713-728-5501
Include $3.00 shipping. COD, VISA and MASTERCARD accepted

CIRCLE 65 ON READER SERVICE CARD

A general purpose programming language for string and list processing and all forms of non-numerical computation.

SNOBOL4+ —the entire SNOBOL4 language with its superb pattern-matching facilities • Strings over 32,000 bytes in length • Integer and floating point using 8087 or supplied emulator • ASCII, binary, sequential, and random-access I/O • Assembly Language interface • Compile new code during program execution • Create SAVE files • Program and data space up to 300K bytes RAM

With **ELIZA** & over 100 sample programs and functions

Have you tried SNOBOL4+?

MUSIC ANALYSIS • TEXT PROCESSING • SYMBOLIC MATHEMATICS • ARTIFICIAL INTELLIGENCE

COMPILER PROTOTYPING • CRYPTOGRAPHY • LINGUISTICS • LIST PROCESSING • GAMES •

For all 8086/88 PC/MS-DOS or CP/M-86 systems, 128K minimum 5¼" DSDD, specify DOS/CPM format

Send check, VISA, M/C to: **$95** plus $3 s/h

Catspaw, Inc.
P.O. Box 1123 • Salida, CO 81201 • 303/539-3884

CIRCLE 9 ON READER SERVICE CARD

eXchanger
CP/M ⟷ ISIS
Emulator

Now move files and programs between your CP/M-80 system and your Intel Series I or II MDS! The ICX package provides complete bidirectional file conversion capability, and even **allows execution** of ISIS-II programs under CP/M using the ISE emulator. The ICX Package is composed of the following two programs:

ICX A Deluxe bidirectional file conversion utility which works with your CP/M system and an 8" floppy drive to provide complete manipulation of an ISIS-II diskette. Takes directories, deletes files, and even initializes a blank disk with the ISIS file structure. Complete C source included. **$89**

ISE An ISIS-II Emulator which allows ISIS programs to run on any CP/M-80 system. Support for all ISIS-II system and monitor calls makes your CP/M micro look like an MDS! Supports banked memory. Complete MAC source included **$89**

Complete ICX Package (ICX & ISE) **$175**

Supplied on Single Density 8" Disk
CP/M Digital Research, Inc. ISIS-II Intel Corp

Western Wares
Box C
Norwood CO 81423
(303) 327-4898

CIRCLE 67 ON READER SERVICE CARD

Interested in writing, reviewing software, or refereeing manuscripts for

COMPUTER LANGUAGE

For information contact:
Craig LaGrow/Editor
131 Townsend St.
San Francisco, CA 94107
(415) 957-9353
BBS#: (415) 957-9370
CompuServe Acct: GO CLM

YOUR CODE MAY BE WASTING ITS TIME!
THE PROFILER™ CAN HELP . . .

- Statistical Execution Profiler
- Works with any language
- Completely configurable
- Up to 16 partitions in RAM/ROM
- Time critical code optimization
- Abnormal code behavior tracking
- Graphic presentation of results
- Easy to use menu interface

THE PROFILER is a software package which gives you, the programmer, a powerful tool for locating time consuming functions in your code and allows you to performance tune your program. With the **THE PROFILER** you can determine where to optimize your code for maximum benefit, then measure the results of your efforts.

Using **THE PROFILER,** you can answer questions like:
Where is my program spending its time?
Why is my program so slow? What is it doing?
Is my progam I/O bound? CPU bound? Are data buffers large enough?
How much improvement did my changes make?

THE PROFILER is completely software based and consists of a system resident driver and a monitor program. The memory partitions can range from 1 byte to 1 megabyte in size and can be anywhere in the address space.

NO ADDITIONAL HARDWARE IS REQUIRED!

Requires an IBM PC or compatible system with a minimum 64k and one drive.

THE PROFILER is available for $175.00 from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

dwb
Associates
P.O. Box 5777
Beaverton, Oregon 97006
(503) 629-9645

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
THE PROFILER is a trademark of DWB Associates.

CIRCLE 20 ON READER SERVICE CARD

EASY To Use!

Developed in England by Southern Software

SBE

Z80 / 8088 (8086 / 80186 / 8087) machine-code development system. With latest Reduced Instruction-Set philosophy.

☐ SBE/TRS 80 (All DOS) $100 + $3 s/h
☐ SBE/PC (PC-DOS/MS-DOS $160 + $3 s/h

Allen Gelder Software
(415) 681-9371
Box 11721 San Francisco, CA 94101

CIRCLE 28 ON READER SERVICE CARD

THE CODE SWAP SHOP

COMPUTER LANGUAGE
SOFTMail
3
CORREOS DE
3

Editor's Note: All programs referred to in this reader-inspired, public domain column will be available for downloading when you call the COMPUTER LANGUAGE *Bulletin Board Service at (415) 957-9370—300/1200 baud—or when you dial into CompuServe and invoke our account by typing "GO CLM".*

[redacted]

A disassembler from New Delhi

All the way from New Delhi, India, comes a disassembler that is more powerful than SID-like disassemblers such as REZ, which have the basic problem of not separating the data area from the instruction area.

Ravindra Kumar Agrawal sends us his own improved version of a disassembler he wrote to solve this problem. "The renaming of labels and the insertion of comments," he said, "can easily be done, and the program's natural aesthetics can be preserved by using a word processor in addition to the disassembler."

On the *COMPUTER LANGUAGE* BBS and on CompuServe are his original source code and the .OBJ file. He also included some DEMO programs and a file DETAILS.DSM, which explains more completely the features of the disassembler.

[redacted]

Accounting system is written with UNIX tools

Richard A. Bilancia, an accountant from Littleton, Colo., wrote in with a program that touches upon the potential application of the relational data base tools included with the UNIX operating system.

The simple accounting system is written in the UNIX shell programming language (the same interactive command interpreter that you use to execute simple UNIX commands) and uses the following UNIX relational data base tools: awk, cat, echo, join, lpr, pr, rm, sed, sed, and sort.

A .DOC file is also presented on the BBS and on CompuServe which will explain the design criteria behind the program itself.

[redacted]

Perform math functions while word processing

The capability to perform mathematical calculations while writing a document enhances the usability of a word processor. For those people with an Eagle computer running Spellbinder, EagleWriter, or Word/125, this may be a very useful tool.

Paul Loughridge Jr. of Kamuela, Hawaii, has written a math function program called **MATHII.WPM** in M-SPEAK, a programming language available with Spellbinder. For $25 plus postage, you can acquire this program on 5¼-in. Eagle II-formatted diskette by writing to P. O. Box 1206, Kamuela, Hawaii 96743. However, the author has also given his permission for us to place it on the BBS and on CompuServe.

[redacted]

Use capsules to create programs

Programming styles can range from highly unstructured to rigid and top down. End users, however, may occasionally wish to actually design a program of their own to do one specific task—e.g., compile a mailing list.

Namir Shammas, Richmond, Va., has designed a way for programmers to provide these end users with a simple tool for letting them expand upon the features of a given software package. The idea is to provide capsule or skeletal abbreviations of more intricate, lengthy programs. In this way, the end users can mimic the design of the semi-pseudocode and go ahead and write their own programs.

To demonstrate how this idea really works, Shammas has provided two sample capsules—one written in Ada, the other in Microsoft BASIC—for *COMPUTER LANGUAGE* readers to consider.

[redacted]

Try the ACTIGRAM approach

The early stages of software development employs careful planning of the data objects to be manipulated and the activities involved. One can map these activities in a way similar to a menu and submenu tree.

Once again, Namir Clement Shammas provides us with the source code to a program called **ACTIGRAM**, in which he demonstrates one method of designing code from the top down. By identifying the macro activities first and the micro activities later, the ACTIGRAM is an example of one person's approach to coding.

Written in Turbo Pascal, this program will allow the user to manage a system of inter-linked activities.

[redacted]

Do you have code for the Swap Shop?

If you've written a program that you'd like to see distributed free of charge to *COMPUTER LANGUAGE* readers, send us a two- to four-paragraph summary of what the program does, how you can make the program electronically available to our magazine (e.g., bulletin board transfer, disk format, CompuServe, etc.), and whether you'd like your name, address, and/or telephone number included in the magazine.

Address all correspondence to: Craig LaGrow, Editor, 131 Townsend St., San Francisco, Calif. 94107. Or call us up on CompuServe or the BBS!

Illustration: Anne Doering

SOFTWARE REVIEWS

DR FORTRAN-77

Hardware required: IBM PC and PC/XT with 192K memory, or any 8086- or 8088-based microcomputer running PC-DOS 2.0+ and CP/M-86.
Price: $350
Available from: Digital Research Inc., 60 Garden Ct., P. O. Box DRI, Monterey, Calif. 93942, (408) 649-3896
Support: 8087 math-coprocessor chip support included, update notices free, minimal charge for bug fixes

In 1965, back when most engineers were still using slide rules, I was introduced to FORTRAN II on an old IBM 1620. Punched cards were the order of the day. The entire instruction set could be written on the inside of a matchbook cover, and anything larger than a trivial program had to be segmented in order to run.

To give you an idea of how primitive FORTRAN II really was, there was no logical *if* construct. The only *if* construct available was an arithmetic *if*:

```
IF (EXPRESSION) LABEL1, LABEL2,
      LABEL3
```

This *if* expression evaluates three values: a positive, non-zero value, a negative value, and 0. If the value is negative, the program execution goes to LABEL1. If 0, it goes to LABEL2. If a positive, non-zero value, it goes to LABEL3.

Fortunately, FORTRAN has changed a lot since those days.

On April 3, 1978, FORTRAN-77 was certified by the American National Standards Institute. This version of FORTRAN is extremely powerful, not only because of its unbelievably potent number-crunching abilities but also because it incorporates string manipulation and the options of structuring the code, using white space and indentation for clarity, and using unformatted I/O.

Digital Research Inc. has now released DR FORTRAN-77, which has been certified as genuine ANSI FORTRAN-77. In order to appreciate how significant and timely this language release is, it would be useful to briefly touch on the potential of FORTRAN under UNIX and explore some of the exciting, new philosophies used in designing computer languages today.

Evidence of FORTRAN's importance as a programming language is that it is one of the three original programming languages native to UNIX. (The other two are C and RATFOR.) Called f77 under UNIX, FORTRAN takes on quite a different flavor under this innovative operating system. Because f77 can call programs written in C, C also calls programs written in f77, and a mix of programs from both languages can be linked.

UNIX gives FORTRAN a new dimension and much more power. Even RATFOR demonstrates FORTRAN's influence. RATFOR (rational FORTRAN) is a language cast in C's image that produces FORTRAN as its output. However, UNIX is becoming such a dominant influence in the computer industry, it is my opinion that FORTRAN's future is assured partly because f77 is part of UNIX.

The influence of UNIX can be seen in the way DR FORTRAN-77 was written by DRI. DRI's method of creating compilers involves a technique that goes back to UNIX, which uses a lexical analyzer (lex) in combination with a compiler compiler (yacc) to define the lexicographic conventions of the language first and create the necessary parse tables according to the syntactic rules of the language second.

The syntax is created in a yacc file, and it follows the conventions of BNF (Backus-Naur Form) grammar. Upon compiling the yacc source code, a C program is output. It too is compiled by cc, and the result is a compiler.

In UNIX, compilers produced in this manner, such as cc and f77, produce a c-intermediate code that is translated into native assembly and linked into executable machine code. The result is language source code that is portable from one UNIX machine to another, regardless of machine size or architecture. Under UNIX, C can call programs written in f77, f77 can call programs written in C, and a mix of programs from both languages can be linked. A C or f77 program written on a IBM PC XT under PCIX (single-user UNIX) can run on an IBM 370 or an Amdahl 580 under UTS UNIX or vice versa.

UNIX gives hardware independence, but the new DRI scheme goes a step beyond and gives operating system independence. The DR FORTRAN-77 language compiler was created as two modules, with a front end and a back end. The front end carries the lexical analyzer (as with UNIX's lex) and the parser. The front end also produces the symbol table.

As a total entity, the front end produces a common intermediate language similar to cc and f77's c-intermediate code. Presently only DR FORTRAN-77 utilizes this method, but other DRI languages are being rewritten in accordance with this scheme. Soon I predict that all DRI languages will produce the same intermediate code.

However, the code generation system goes much deeper than a common intermediate language. Code optimizing schemes are part of the picture as well. DRI has opted to use postfix notation (reverse polish) when it converts mathematical expressions.

If you have ever had to deal with a Hewlett-Packard calculator, you have been exposed to reverse polish. Reverse polish requires you to enter everything in reverse, more or less. The reason is to put both the values and the operators in "stack order." For example, 2 + 3 is entered as 2 3 +, because numbers must be seen before operators as they are taken off the stack.

A great deal of effort has gone into standardizing the front ends. All the DRI languages use the same IEEE data types. All use the same passing conventions as well. This effort produces a series of languages that can interact with each other. DR FORTRAN-77 will be able to call routines written in C, PL/I-86, CBASIC or Pascal MT+.

Presently, however, the cross calling is limited to FORTRAN and C. Since each language has its own unique virtues and there is no such thing as one perfect, do-all language, the ability to cross call routines allows the programmer to write routines in languages best suited for the operation at hand. A good example is using FORTRAN for the numeric processing and calling C to do the "bit-diddling" (bit manipulation).

If DR FORTRAN-77 and the other DRI languages are to work with each other, they must deal with the same data types. The names do not have to be the same but the data types must. DRI has isolated 19 data types used in its language implementations. None of DRI's languages use them all, but PL/I comes close. All data types are IEEE defined (IEEE short, long, etc.). As a result, they can be passed from module to module without language convention restriction.

The magic does not stop with the language front ends. Common, universal back ends are used as well. Whereas the front ends bring separate languages into a common intermediate language for uniformity, the back ends generate code and search run-time libraries for different processors (and systems). As a result, the same intermediate code can produce object code for the 8086, 8088, 80286 and the 68000 processors. Figure 1 illustrates the concept of front and back ends in compiler design.

DRI portable compiler design

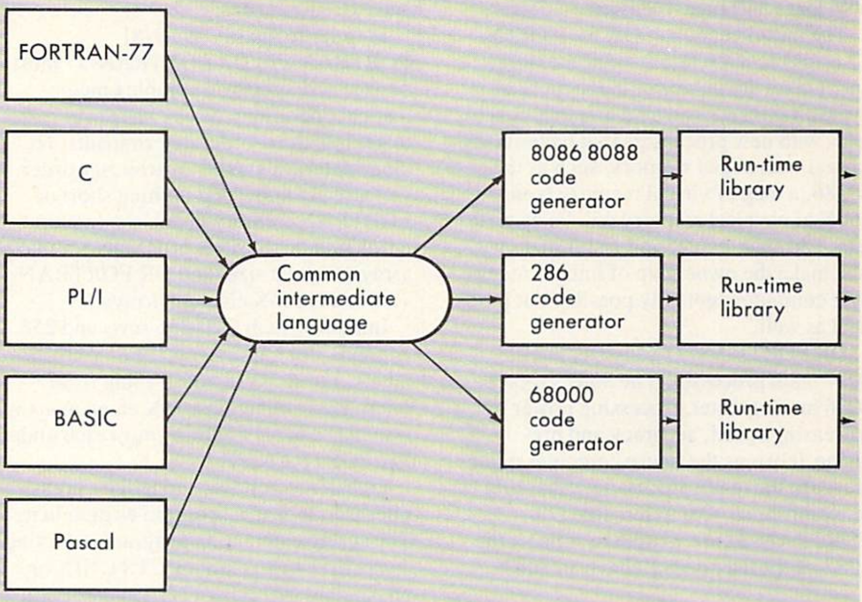


Figure 1.

In time, the common intermediate language/universal back end system will extend across the range of common processors and non-hardware-specific operating systems. The implications are incredible: several different programming languages, all capable of talking to each other and generating code for CP/M, MP/M, Concurrent PC-DOS, MS-DOS, PC-DOS, and UNIX for the 8086, 8088, 80286, and 68000 family of processors.

If DR FORTRAN-77 had to be summed up in one sentence, that sentence would be:
DR FORTRAN-77 is full ANSI X3.9-1978 FORTRAN.

This implementation is the full set. The name DR FORTRAN-77 also implies that the language should be in accordance with the UNIX definition of the language as outlined by S.I. Feldman and P.J. Weinbeger in *UNIX Tutorial*, vol. 4. DR FORTRAN-77 is all of these and more. A little has been added, but nothing has been left out.

Like UNIX's f77, DRI's DR FORTRAN-77 keeps Holerith notation (an anachronism of anachronisms) to retain upward compatibility all the way back to FORTRAN II. Even the rusty old arithmetic *if* and *GOTO* have been left intact. All data types are included, including complex—the result of the square root of a negative number combined with a real number.

FORTRAN used to be the world's worst string handler. However, like FORTRAN-77, DR FORTRAN-77 can deal with character data very well. It also has a number of string functions. All data types are available, including extended to take advantage of the 8087's monumental ability to handle huge numbers with blinding speed.

DR FORTRAN-77's enhancements include 40-character variable names, including the underscore character and the dollar sign. This results in ultra-descriptive variable names like:

```
maximum_stress
```

and

```
undepreciated_usvalue_after_taxes
```

These descriptive names allow assignments of expressions like:

```
stress = (8* o_d - wire_dia)* K * load
    / (pi * wire_dia**3)
```

For those of us who remember old versions of BASIC that allowed two-letter variable names, maximum, this is a gigantic step forward. Code used to be hard to read without a variable dictionary. DR FORTRAN-77 is one of several modern programming languages that is helping us get away from all that.

The other side of the DR FORTRAN-77 system is the linker. Don't expect anything like *flink* or *forlink*. The linker is DRI's standard LINK-86. DRI's programming utilities, such as MAC, RMAC, XREF, LIB-86, etc., are the same for all their languages. That is how DR FORTRAN-77 calls C and C calls DR FORTRAN-77.

Another benefit of DR FORTRAN-77 is the ability to overlay to mind-boggling nesting depths. Because memory is getting cheaper and because paged memory (virtual memory) is becoming more commonplace, the need for using overlays will gradually diminish. Until then, if your program exceeds the memory space of the machine and you do not have demand-paged virtual memory, you have to overlay. DRI linkers have always had the ability to perform complex overlays with relative ease.

Therefore, if overlaying is thrust upon you out of necessity, DRI languages using LINK-86 (C, PL/I-86, DR FORTRAN-77) ease this task considerably. One effective method is using a menu-driven program as the overlay root to call separate program modules as overlays. In this way, exquisite menu-driven trees can be constructed. The structure of the program then becomes self-imposing and the chore of using overlays is made that much easier to deal with.

Using overlays helps you put your software on a piece of hardware that otherwise couldn't be used. (If you think overlaying is a chore, in the old days we used to have to segment FORTRAN programs using card decks to pass data from one execution to another in batch mode.)

Many people buy a 16-bit language compiler and automatically assume that it is capable of creating code that can effectively address all of the memory on a 16-bit machine. Nothing could be further from the truth.

Unless a language compiler is capable of creating programs with memory models, it is not using memory efficiently. C, DR FORTRAN-77, and PL/I-86 create programs with memory models, and that feature alone sets them in a special class. DR FORTRAN-77 creates programs with both large and small memory models.

The initial version of DR FORTRAN-77 is for the 8086 family of processors, and it takes advantage of much more of the available, addressable memory space. Now with new processors that have ability to deal with paged memory, such as the 80286, a meg of virtual memory from 256K of physical memory will be no problem. Cheaper, cooler and faster memory will make the ownership of integer megabyte computers not only possible but practical as well.

DR FORTRAN-77 supports Intel's 8087 math processor. The 8087 gives the 8086 much greater processing power by increasing speed, accuracy and precision.It brings the micro computer much closer to the math-processing capability of mainframes. Compilers like DR FORTRAN-77 are written to support the 8087. If the user doesn't have an 8087, routines are also written to emulate the 8087 math processor, giving the user the same increased ability, without the speed and extreme accuracy.

In order to qualify as ANSI FORTRAN-77, DR FORTRAN-77 must be able to do perform complex mathematical operations. If you have ever had to get into third-order determinants, let alone determinants of an arbitrary order, you already know that nothing short of FORTRAN's ability to do array manipulation will do. Serious programs get into arrays of great size, and DR FORTRAN-77's limit is 64K element arrays.

Imagine a matrix of 256 rows and 256 columns, for a grand total of 65,536 elements. For those of you coming from micro environments, a 64K element array is unheard of. It's within your reach under DR FORTRAN-77.

One of the best ways to judge a compiler is to look at a program written in it. The spring calculation program in Listing 1 was first written under UTS UNIX on an Amdahl 470 using f77. This program makes a call to an external routine written in C to clear the screen. This may sound like extra work, but linking a C program to an f77 program is no more work than linking an f77 program to an f77 program.

The following is the C program:

```
/*
        clear.c
*/
clear_()
{
        system (''ascii ff'');
}
/**/
```

The spring calculation program (spring.f) is written in f77, but it is also top-down, structured, and modular. The compact main block calls the three blocks that comprise the action portion of the program. The main block is trapped in a "do-nearly-forever" because calculating springs is a highly iterative process. By

```
      program spring
c ------------------------------------------------------------------
c   Spring program to calculate helical compression springs
c
c   Bruce H. Hunter    July 24, 1984
c
c   Note : link with clear.c
c --------------------------------------------------------------------
      PARAMETER (maxint = 65535)
      INTEGER   itr
      external clear

      CALL clear
      do 10 itr = 1, maxint
         PRINT *, ' '
         PRINT *, 'Helical compression spring program'
         PRINT *, ' '
         CALL input
         CALL calc
         CALL output
10   CONTINUE
     END

     SUBROUTINE input

     COMMON /incalc/
   $     g, wiredia, ncoils, load, height, od
     DOUBLE PRECISION
   $     g, wiredia, ncoils, load, height, od

        PRINT *, 'input wire dia, od'
        READ  *, wiredia, od
        PRINT *, 'number of active coils'
        READ  *, ncoils
        PRINT *, 'torsional modulus'
```

Listing 1 *(Continued on following page).*

```
        READ  *, g
        PRINT *, 'specified load at specified height'
        READ  *, load, height
      END

      SUBROUTINE calc

      PARAMETER (pi = 3.1415927)
      COMMON /incalc/
    $     g, wiredia, ncoils, load, height, od
      COMMON /calcout/
    $     rate, freelen, mstress, coils, stress
      DOUBLE PRECISION
    $     g, wiredia, ncoils, load, height, od, stress,
    $     c, k, rate, freelen, sumtrav, maxload, mstress, coils
      INTEGER inputno

c       Wahl Factor c
      c = (od - wiredia) / wiredia
      k = (4.0 * c - 1) /(4.0 * c - 4) + 0.613 / c
      stress = (8.0 * od - wiredia) * k * load / (pi * wiredia**3)
      IF ( (stress) .GT. (200000.0) ) THEN
          WRITE (6, 140) stress
          PRINT *, 'enter 1 to continue 0 to restart'
          READ  *, inputno
          IF ( (inputno) .EQ. (0) ) GO TO 999
      END IF
      rate = g * wiredia**4 / (8.0 * (od - wiredia)**3 * ncoils)
      freelen = load / rate + height
      coils = ncoils + 2.0
      sumtrav = freelen - coils * wiredia
      maxload = sumtrav * rate
      mstress = 8.0 * (od - wiredia) * k * maxload / (pi * wiredia**3)
140   FORMAT ('stress = ', f9.0)
999   END


      SUBROUTINE output

      COMMON /calcout/
    $      rate, freelen, mstress, coils, stress
      DOUBLE PRECISION
    $      rate, freelen, mstress, coils, stress

      INTEGER inputno
      WRITE (6, 100) stress
      WRITE (6, 110) coils
      WRITE (6, 120) rate, freelen
      WRITE (6, 130) mstress
      PRINT *, 'enter 0 to exit'
      READ  *, inputno
      IF ( (inputno) .EQ. (0) ) STOP

100   FORMAT ( 'stress at working height ', f9.0)
110   FORMAT ( 'total coils  ', f5.1)
120   FORMAT ( 'rate  ', f9.3, 'free length  ',f8.3)
130   FORMAT ( 'stress at closed height  ', f9.0)
      END
```

Listing 1 *(Continued from preceding page).*

```
      program spring
c -------------------------------------------------------------------
c   Spring program to calculate helical compression springs
c
c   Bruce H. Hunter    July 24, 1984
c
c   Note : link with clear.c
c --------------------------------------------------------------------
      PARAMETER (maxint = 65535)
      INTEGER   itr
      external clear

      CALL clear
      do 10 itr = 1, maxint
         PRINT *, ' '
         PRINT *, 'Helical compression spring program'
         PRINT *, ' '
         CALL input
         CALL calc
         CALL output
10   CONTINUE
     END

     SUBROUTINE input

     COMMON /incalc/
   $      g, wire_dia, n_coils, load, height, o_d
     DOUBLE PRECISION
   $      g, wire_dia, n_coils, load, height, o_d

        PRINT *, 'input wire dia, o_d'
        READ  *, wire_dia, o_d
        PRINT *, 'number of active coils'
        READ  *, n_coils
        PRINT *, 'torsional modulus'
        READ  *, g
        PRINT *, 'specified load at specified height'
        READ  *, load, height
      END

      SUBROUTINE calc

      PARAMETER (pi = 3.1415927)
      COMMON /incalc/
    $     g, wire_dia, n_coils, load, height, o_d
      COMMON /calcout/
    $     rate, free_length, max_stress, coils, stress
      DOUBLE PRECISION
    $     g, wire_dia, n_coils, load, height, o_d, stress,
    $     c, k, rate, free_length, sum_travel, max_load, max_stress, coils
      INTEGER input_no

c       Wahl Factor c
      c = (o_d - wire_dia) / wire_dia
      k = (4.0 * c - 1) /(4.0 * c - 4) + 0.613 / c
      stress = (8.0 * o_d - wire_dia) * k * load / (pi * wire_dia**3)
      IF ( (stress) .GT. (200000.0) ) THEN
          WRITE (6, 140) stress
```

Listing 2 *(Continued on following page).*

```
        PRINT *, 'enter 1 to continue 0 to restart'
        READ  *, input_no
        IF ( (input_no) .EQ. (0) ) GO TO 999
      END IF
      rate = g * wire_dia**4 / (8.0 * (o_d - wire_dia)**3 * n_coils)
      free_length = load / rate + height
      coils = n_coils + 2.0
      sum_travel = free_length - coils * wire_dia
      max_load = sum_travel * rate
      max_stress = 8.0 * (o_d - wire_dia) * k * max_load / (pi * wire_dia**3)
140   FORMAT ('stress = ', f9.0)
999   END


      SUBROUTINE output

      COMMON /calcout/
     $     rate, free_length, max_stress, coils, stress
      DOUBLE PRECISION
     $     rate, free_length, max_stress, coils, stress

      INTEGER input_no
      WRITE (6, 100) stress
      WRITE (6, 110) coils
      WRITE (6, 120) rate, free_length
      WRITE (6, 130) max_stress
      PRINT *, 'enter 0 to exit'
      READ  *, input_no
      IF ( (input_no) .EQ. (0) ) STOP

100   FORMAT ( 'stress at working height ', f9.0)
110   FORMAT ( 'total coils  ', f5.1)
120   FORMAT ( 'rate  ', f9.3, 'free length  ',f8.3)
130   FORMAT ( 'stress at closed height  ', f9.0)
      END
```

Listing 2 *(Continued from preceding page).*

hand, or even with a calculator, these calculations can consume an entire day. Using a program to do the work reduces the process to a few minutes.

The first subroutine in Listing 1 is the input routine. It uses free format input and output. Variables do not have to be declared, but I am a structured programmer at heart, and I just can't let variables default to integer and real, or worse yet, go unrecognized by anyone reading the source code before they are encountered in the program block. A common statement at the top of the block makes all the variables input common to the input and calculation blocks. Notice the white space, free indentation, and the lack of line numbers and labels. The dollar sign is a continuation character. Any reasonable character can be used as long as it is in the sixth column.

The second subroutine is the calculation block. Here is FORTRAN doing what it does best, grinding large and small numbers into fine dust. Few languages have more forms of the *if* statement than FORTRAN. The *if* in this block is a block form *if*, trapping all statements between the *then* and the *end if*. The *else* and *else if* are perfectly legitimate and can be nested to extreme depths.

This subroutine has two common statements, *incalc* and *calcout*, to reflect the two subroutines they service. The only possible weakness in the common is the inability to have the same variable in three different commons. The variable *ncoils* had to be changed to *coils* to have it shared in three blocks.

The final subroutine is the output routine. It utilizes the older formatted write statements to present the data in the best and most readable form, a strong advantage to any language that supports it.(PL/I also has formatted and unformatted I/O.)

As mentioned earlier, DRI's FORTRAN implementation includes all of the FORTRAN ANSI 77 and f77 features and a few more. f77 supports eight-character variable names but no underscore characters or imbedded dollar signs. DR FORTRAN-77 supports 40-character variable names as well as imbedded underscore characters and dollar signs.

Now look at Listing 2. In this version of the spring program, written for DR FORTRAN-77, notice how much easier it is to read and follow because of the increased length of the variable names and the use of the underscore character.

In conclusion, DR FORTRAN-77 is no "me-too" project. It is a major step in portable languages geared to multiprocessor, multi-operating system, multilanguage interface applications. We will see it under PC-DOS, CP/M, and eventually UNIX, running on the 8086, 8088 and 80286 as well as the 68000 family of processors.

Today there is a need for a family of languages that can address all three of these operating systems as well as emerging systems that will have UNIX with PC-DOS emulation. DRI has nearly accomplished this task, and DR FORTRAN-77 is their first offering in the new generation of multi-OS, multiprocessor transportable languages.

By Bruce Hunter

mbp COBOL

Hardware required: IBM PC, XT, or AT
Price: $750
Available from: mbp Software & Systems Technology, 7700 Edgewater Dr., Suite 360, Oakland, Calif. 94621, (415) 632-1555.

You can discuss many things that are sure to start an argument. Religion and politics come to mind first, but I would like to add a third: the choosing of a programming language.

This review is not intended to convert Pascal or C programmers over to COBOL but is meant to show you why mbp COBOL is currently the Cadillac COBOL compiler on the microcomputer market.

First, no COBOL compiler now on the market approaches mbp COBOL for speed of execution. (Table 1 shows some timings of program execution.)

However, while speed is indeed important, it is not the only piece of the pie. Ease of use, support, and documentation play an equally important role in which computer language you pick to complete a given task. This review deals with the features and characteristics of the mbp compiler that might have gone unnoticed if we were to deal in terms of speed of execution alone.

mbp COBOL has compromised the software developer's position for speed of execution. What this means is that while the programs do indeed run very, very fast, the programmer is saddled with the overhead that makes this speed possible. Nothing is wrong with this approach but, speaking as a programmer, I would have liked to have had a faster compiler and retained speed of execution.

Execution speed comparisons (in seconds)

Function performed	MBP on IBM-XT	Microsoft on IBM-XT	Microsoft on Zenith Z100	Microsoft[1] on Zenith Z100
10,000 performs	1	4	7	4
10,000 goto's	1	9	14	8
10,000 adds and subtracts	19	59	65	37
Add integers 32,767 times	4	162	192	111
1,000 moves	3	14	10	6
10,000 If statements	3	15	15	9
Concatenate a string 10,000 times	7	71	71	46
Read a 100 element table 100 times by indexing	1	95	108	62
Read a 100 element table 100 times by subscripting	19	76	86	49
Sieve of Eratosthenes for 2 iterations	153	712[2]	704	403

1. Z100 running at 7.5 meg clock speed vs. 4.77 meg for standard Z100 machine.
2. Version 1.12 of MS-COBOL now allows SIEVE program to compile.

Table 1.

Object Module Comparison

Program Name	MBP compiled size (bytes)	Microsoft compiled size (bytes)	Difference	Percent
ISAMTEST	30,208	21,120	9,088	30.08
GIBSON	46,720	33,792	12,928	27.67
PCPERF	12,544	8,320	4,224	33.67
PCGOTO	12,672	8,320	4,352	34.34
PCADDSUB	12,800	8,320	4,480	35.00
PCMLTDIV	12,800	8,320	4,480	35.00
PCMOVE	13,568	8,064	5,504	40.57
PCSTRING	13,056	7,936	5,120	39.22
PCILOOK	12,928	8,576	4,352	33.66
PCSLOOK	14,208	8,576	5,632	39.64
SIEVE	22,784	16,896	5,888	25.84

Since both systems use the same linker, namely the LINK program that comes with the IBM or Zenith system, the difference is entirely within the compilers themselves.

Table 2.

As you saw from Table 1, the compiler produces very fast execution modules. I re-compiled the sample programs under the Microsoft COBOL compiler version 1.12. While the Microsoft compiler produces smaller modules—from 25% to 40% smaller—they execute rather slowly when compared to mbp COBOL (Table 2). The size of the Microsoft modules do not take into account the size of the RUNCOB.EXE module since it only has to be on the disk once.

As I mentioned earlier, speed is nice, but in the real-world environment, most programs will be waiting on operator input from 80% to 90% of the time. Execution speed, although important, is further down the list of desirable features in a COBOL compiler—unless you're running a stand-alone calculation program.

While it is true that the mbp compiler requires 1.5MB of disk space to compile and approximately 200K to 512K of additional space for work files, most software developers are running on a hard disk system anyway. The vendor does state that you can compile using just two floppy disks, but I don't know of anyone who would be thrilled at the idea of spending 15 to 30 min changing floppy disks.

If you are doing much development on fairly large systems, you may want to have at least 20MB to save headaches. If you are running on a 10MB system, then you should back up those files that are not required and try to free up at least 5MB of space for software development.

Aside from the rather large disk space requirements, the compiler was only about 50% slower than the Microsoft compiler. I took the SIEVE of Eratosthenes program and compiled it with the object listing option turned on as well as cross-reference and storage map options. The output was sent to the hard disk instead of the printer. Total compile time was 5.5 min.

I then compiled the program under MS-COBOL, and it required 2.1 min to compile. Remember, this is not necessarily a fair test since Microsoft does not have any options for a cross-reference or object listing.

From a developer's point of view, the output of the mbp compiler approaches that of a mainframe system. The first page of output shows you the source, object, list file, work file, and all the options you had working at the time of compile. The storage map and the cross-reference listing alone make this compiler worth the price.

While not required in smaller programs, a cross-reference listing can save many hours when looking for a data name to change a program you wrote many months ago. As you can see in Listing 1, the cross-reference listing is easy to use and can be turned on or off by an option selection.

With the latest version of this compiler (version 7.4), mbp has implemented a COBOL sort as well as provided chaining capability. The sort implementation is not the standard sort implementation as defined in COBOL. The familiar *SORT* verb

```
SORT SORTWORK ON ACSENDING
     KEY. SORT-NAME,
          SORT-AR
          USING SORTIN,
          GIVING SORTOUT.
```

would appear quite different when used from within an mbp COBOL program. In this compiler, you call the sort as a subroutine, for example:

```
CALL "MBPSORT" USING INPUT-
     FILE,
          OUTPUT-FILE,
          CONTROL-STATEMENT,
          SORT-STATUS.
```

The CONTROL-STATEMENT data name defines a 30-character alphanumeric field that contains the specifics of how you want the sort executed. While a minor inconvenience is imposed by not being a standard sort verb usage, the mbp sort is very fast. It sorted 1,000, 128-byte records in approximately 7 sec.

The *CHAIN* verb is also a callable routine that allows you to pass and receive parameters between programs. Parameters are passed as part of the *CHAIN* statement and are not required to be set up in the linkage section of the program.

The test of any new compiler from a programmer's point of view is based on how easy is it to use and whether questions can be answered via the supporting documentation. mbp COBOL gets flying marks in both these areas.

The manual, which fills a 2-in, 3-ring binder, is not only well organized; it also has menu tabs that help you easily find specific sections of interest without a lot of page searching. The compiler arrives on five diskettes, and a sample program disk shows examples of the *SORT* and *CHAIN* features. Also included is a sample program of the interesting way in which mbp COBOL handles screen formatting.

Using the Screen Management System (SMS) brought back shades of CICS screen design. For those unfamiliar with CICS, let me just say that CICS has implemented a screen design system that allows your screen to be a separate module not resident in the actual program, as under Microsoft COBOL.

Well how does the mbp compiler fair in the long run? I'd rate it excellent in speed, from a user's point of view, and fair in performance, from a programmer's point of view. While I would like to see it compile faster and produce optimized object code, you just can't have everything. In the next mbp COBOL version, to be released during the first quarter of 1985, some of the current shortcomings should be corrected.

Included in the initial price of the software is a licensing portion that allows you to produce up to 50 applications before incurring additional royalty charges. You can opt to pay the vendor $2,000 for an unlimited distribution of modules that you would create using their compiler, but I find this a little steep for most software distributors. The current update policy specifies that you must pay the difference between the amount that you paid for the compiler and the current selling price for the latest version.

If you are looking for an excellent compiler offering user speed that is second to none, look no further. mbp COBOL does use a fair amount of disk space—but you always wanted to upgrade to that 20MB drive anyway. I am willing to sacrifice the extra disk space and slightly increased compile time for the features provided by the mbp compiler. They are indeed well worth the time and effort.

By Chuck Ballinger

```
 SOURCE    DATA/PROCEDURE NAME              REFERENCED BY STATEMENTS
  LINE

    73   CALCULATE-ELAPSED-TIME               46
    66   COMPARE-EXIT                         53    60
    59   COMPARE-ROUTINE                      53
    37   DISPLAY-MESSAGE                    NOT REFERENCED
    27   ELAP-TIME                            86    88
    21   FLAG-AREA                          NOT REFERENCED
    22   FLAGS                                57    60    70
    16   I                                    51    52    53    54    57
                                              60    61    61    62
    49   ITERATION-ROUTINE                    44
    17   K                                    62    63    70    71
    15   MISC                               NOT REFERENCED
    19   PRIME                                61    62    71
    18   PRIME-COUNT                          50    64    89
    25   START-TIME                           43    74    76    77    86
    26   STOP-TIME                            45    75    81    82    86
    69   STRIKOUT                             63
    56   TABLE-FILL-ROUTINE                   51
    24   TEST-TIMES                         NOT REFERENCED
    42   TESTING-MODULE                     NOT REFERENCED
    30   TST-HRS                              80    85
    31   TST-MINS                             79    84
    33   TST-MSCS                             77    82
    32   TST-SECS                             78    83
    29   TST-SUB-TIME                         76    81
```

Listing 1.

ZCPR3

Hardware required: Z80, 8080, 8085, NSC-800 machines
Price: $39 to $180 (depending upon utilities purchased)
Available from: Echelon Inc., 101 First St., Los Altos, Calif. 94022, (415) 948-3820
Support: User support through telephone and written response

Over the years, since the origin of the CP/M operating system, members of various CP/M user groups have attempted to improve upon and ocrrect what some felt were major shortcomings.

One of the most noted of these attempts was a program called ZCPR (Z80 Command Processor Replacement). Just as its name implies, ZCPR was designed to replace the console command processor portion of CP/M. (For those readers not familiar with the CP/M architecture, CP/M is made up of three basic parts: CCP, the console command processor; BDOS, the basic disk operating system; and BIOS, the basic I/O system.)

Since it was first developed in 1980, ZCPR has gone through many revisions. ZCPR3 is the latest and by far the most powerful version of ZCPR now available. All of the features of ZCPR2 have been retained and many new ones added. ZCPR2 may have given us a hint of what was to come in that it had the bare bones beginning of a modular configuration.

ZCPR3 is a completely modular system. Once the ZCPR3 package has been installed (more about that later), desired modules can be modified and loaded at any time.

Figure 1 shows the ZCPR3 memory image of the full featured system I installed on my test system, a 64K Z80 CCS S-100 system with a Soroc terminal.

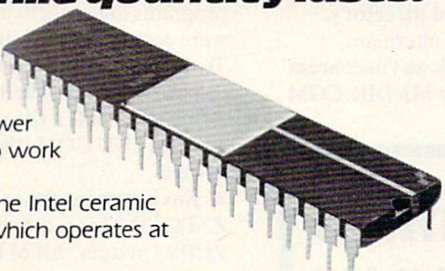
```
  Address

   FFFF  -----------------------------------------------------------
          |          ZCPR3 Input/Output package (IOP)              | 2.5K
   F600  -----------------------------------------------------------
          |          ZCPR3 External Path                           |  16 bytes
   F5F0  -----------------------------------------------------------
          |          ZCPR3 External Stack                          |  48 bytes
   F5C0  -----------------------------------------------------------
          |          ZCPR3 Command Line Buffer                     | 192 bytes
   F500  -----------------------------------------------------------
          |          ZCPR3 Memory-Based Named Directory (S)        | 256 bytes
   F400  -----------------------------------------------------------
          |          ZCPR3 External File Control Block             |  48 bytes
   F3D0  -----------------------------------------------------------
          |          ZCPR3 Message Buffers                         |  80 bytes
   F380  -----------------------------------------------------------
          |          ZCPR3 Shell Stack                             | 128 bytes
   F300  -----------------------------------------------------------
          |          ZCPR3                 |         Z3TCAP (S)    | 128 bytes
   F280   |              Environment       -------------------------
          |                  Descriptor (S)                        | 128 bytes
   F200  -----------------------------------------------------------
          |          ZCPR3 Flow Command Package (S)  (FCP)         | 1K
   EE00  -----------------------------------------------------------
          |          ZCPR3 Resident Command Package (S)  (RCP)     | 2.5K
   E400  -----------------------------------------------------------
          |          CCS300BIOS with modified Cold Boot Routine    |
          |                   to Initialize All Elements of the    | 3.8K
          |                   ZCPR3 System Above                   |
   D500  -----------------------------------------------------------
          |          BDOS                                          | 3.5K
   C700  -----------------------------------------------------------
          |          ZCPR3 Command Processor                       | 2K
   BF00  -----------------------------------------------------------
          |          Transient                                     |
          |                Program                                 | 48K
          |                     Area                               |
    100  -----------------------------------------------------------
          |          BDOS and ZCPR3 Buffers                        | 256 bytes
      0  -----------------------------------------------------------
```

Figure 1.

8087 SALE

$150 while quantity lasts.



Put the speed and power of the 8087 to work for you.

We've got the Intel ceramic 8087-3 chip which operates at 5 MHz.

Order by phone and we ship the same day for Visa or MasterCard customers. Or you can send cash, check or M.O. (Sorry, no COD's). Add California state tax, if applicable and $3 for shipping in the U.S.A. or Canada; $15 for foreign air mail. **(415) 827-4321**

VISA MASTERCARD

Steve Rank, Inc.
1260 Monument Blvd., Concord, CA 94518

CIRCLE 37 ON READER SERVICE CARD

WRITE

The Writer's Really Incredible Text Editor lives up to its name! It's designed for creative and report writing and carefully protects your text. Includes many features missing from WordStar, such as sorted directory listings, fast scrolling, and trial printing to the screen. All editing commands are single-letter and easily changed. Detailed manual included. Dealer inquiries invited. WRITE is $239.00.

BDS's C Compiler

This is the compiler you need for learning the C language and for writing utilities and programs of all sizes and complexities. We offer version 1.5a, which comes with a symbolic debugger and example programs. Our price is (postpaid) $130.00.

Tandon Spare Parts Kits

One door latch included, only $32.50.
With two door latches $37.50.
Door latches sold separately for $7.00.

All US orders are postpaid. We ship from stock on many formats, including: 8", Apple, Osborne, KayPro, Otrona, Epson, Morrow, Lobo, Zenith, Xerox. Please request our new catalog. We welcome COD orders.

Workman & Associates
112 Marion Avenue
Pasadena, CA 91106
(818) 796-4401

WORKMAN & ASSOCIATES USER-FRIENDLY SOFTWARE

CIRCLE 68 ON READER SERVICE CARD

PROGRAMMER'S DEVELOPMENT TOOLS

IBM Personal Computer Language and Utility Specialists

LANGUAGES:	List	Ours
Lattice C Compiler	$500	**295**
STSC APL*Plus/PC *Sale Priced*	595	**469**
DeSmet C Compiler with Debugger	159	**145**
CB-86 by DRI	600	**429**
Instant-C by Rational Systems, *Interpretive C*	500	**469**
8088 Assembler w/Z-80 Translator 2500 AD	100	**89**
C Programming System by Mark Williams	500	**459**

Call for Prices and Information about other Languages.

Special Holiday Season Sale Price!
Computer Innovations C-86 Compiler $278
Performance, Features and Low Price, make the C.I. C-86 a Holiday Season Value. Save over $30 from our normal price of $309. Call for more information and details.

UTILITIES:

******C Functions Library Sale******

C Utility Library for C-86 and Lattice New from Essential Software Written 99% in C	$149	**119**
The **Greenleaf Functions** for C-86, Lattice, and Mark Williams C Compilers	175	**139**

Each product features a full library of over 200+ C Functions. No Royalties. Both include source code.

Communications Library by Greenleaf *New*	160	**139**
Btrieve by SoftCraft	245	**199**
C-Food Smorgasbord	150	**109**
Trace-86 by Morgan Computing	125	**115**
OPT-TECH Sort *High Performance Utility*	99	**87**
C Power Paks from Software Horizons	CALL	**CALL**
Phact by Phact Associates	250	**199**
Plink-86 *Overlay Linkage Editor*	395	**310**
Panel *Screen Design/Editing* by Roundhill	350	**234**
Profiler by DWB & Associates	125	**99**
Halo *Color Graphics for Lattice*, CI-86	200	**125**
GraphiC from Scientific Endeavors	195	**179**
Windows For C by Creative Solutions	150	**109**

A SOLID GOLD VALUE
CodeSmith-86 Debugger
Version 1.8 by Visual Age
Retail $145, Our Normal Price $129
Special Sale Price! $109
Sale Price effective until 11/23/84.

Prices are subject to change without notice

Call for our New Catalog consisting of 200+ Programmer's Development Tools Exclusively for IBM PC's and Compatibles.

Account is charged when order is shipped.

VISA **1-800-336-1166** MasterCard

Programmer's Connection
281 Martinel Drive
Kent, Ohio 44240
(216) 678-4301 (In Ohio)

"Programmers Serving Programmers"

CIRCLE 54 ON READER SERVICE CARD

The ZCPR3 modules can be in almost any order you desire as long as they are located above BIOS. As seen in Figure 1, the trade-off for the implementation of these features is a reduced TPA (transient program area).

The following is a break down of the ZCPR3 modules and buffers:

- **IOP (Input/Output package).** As in ZCPR2, this is a user supplied, redirectable I/O module. An example is provided in the ZCPR3 package. This module must be written by the user as it is hardware dependent. The utility LDR.COM is used to load this module and the utilities RECORD.COM, DEVICE.COM, and DEV.COM are used to manipulate I/O redirection.
- **External path buffer.** This buffer is as described for ZCPR2. The utility PATH.COM is used to set up the desired search paths.
- **External stack buffer.** This is an optional space saver for ZCPR3. By keeping the stack external, more features can be included.
- **Command line buffer.** Many of the ZCPR3 utilities take advantage of this buffer to initiate batch processes, in particular, the command file processors ZEX.COM and SUB.COM.
- **Memory-based named directory.** Named directories are an alternate method of identifying disk and user areas under ZCPR3. The utility MKDIR.COM is used to create named directory files, and LDR.COM is used to load them. PWD.COM displays the active named directories, and CD.COM is used to move from one named directory to another.
- **External file control block.** As with the external stack, it is a space saver.
- **Message buffer.** Many of the ZCPR3 utilities have the capability of passing information between each other. This is their communication buffer.
- **Shell stack.** Several of the ZCPR3 utilities are actually shells that replace ZCPR3 while they are running. The shells have the ability of allowing the user to run programs under them just as if the user were communicating directly with CP/M. To do this, an external stack area is required. Some of shell utilities are VFILER.COM, a file utility program, and VMENU.COM, the ZCPR3 menu system.
- **Environment descriptors and Z3TCAP.** This is the road map of the ZCPR3 system. All of the ZCPR3 utilities use this module to determine the configuration of the ZCPR3 system. The environment descriptors contain pointers to and information about all of the other ZCPR3 modules. Z3TCAP is the terminal definition portion of the descriptors. Many of the utilities will take advantage of your terminal's capabilities if this module is correctly installed. Both modules are loaded with LDR.COM
- **FCP (flow control package).** This package, when used with the command file processor ZEX.COM and the utility GOTO.COM, form a powerful batch processing facility. Many different conditional tests can be performed by the FCP. For example, a test can be made to determine the existence of a file. If the file is present the batch processor could perform some operation on it or if it doesn't exist the processor could abort or go on to another operation. The *GOTO* utility allows jumping forward or backward to labels within the batch stream.
- **RCP (resident command package).** Because the CCP replacement must reside within a fixed boundary range, the number of additional features and commands that can be added is limited. To get around this problem ZCPR3 uses an externally located module called the RCP to extend the capabilities of the CCP replacement.
- **Modified BIOS.** This is a user modified version of the standard BIOS provided with your system. The cold boot routine in your BIOS must be modified to initialize the areas of memory used by the ZCPR3 buffers and packages. If the IOP redirectable I/O package is to be implemented, the BIOS jump table will also require modification. Some of the BIOS calls are redirected to the IOP.

The ZCPR3 package received for this review consisted of 10 single-sided, single-density 8-in disks and a preliminary sampler manual that included installation instructions and examples.

3,000 Programmers depend on us
to find, compare, evaluate products and for *solid value.*

THE PROGRAMMER'S SHOP serves serious microcomputer programmers . . . from giant institutions to small independents. *Specializing* helps us provide 100s of programming products . . . technical literature . . . specialized evaluations and more to help you find and evaluate. Other services like . . . special formats . . . rush delivery . . . payment options (POs, COD, credit cards, etc.) . . . newsletters . . . and reports *help you save time, money, and frustration and get* ***solid value.***

Intriguing New Products

BRIEF™ THE PROGRAMMER'S EDITOR for PCDOS is "Out of the way", fast, windows, undo, macros $195

HS/FORTH - fits professionals with great doc, MSDOS interface, full RAM, ASM, graphics, more. Consider a solid FORTH. $210

"BASICA COMPILER", also access all RAM, modules, structured. BetterBASIC, PCDOS $195

For CP/M-80

ECOsoft C is now complete, rich, fast, has library source, trig $225

Edit programs with VEDIT ($119), MINCE ($149) or "C"SE with source ($75)

Other Key Products

C86 by CI ($339), Lattice ($359) from Lifeboat or Microsoft, and Williams C ($475) are in a tight battle. Which is best for integration with Fortran? 8087? support libraries? speed? debugging?

FORTRAN-86 from Microsoft ($259) is improving with libraries for graphics ($175), screen ($265).

LISP by Integral Quality ($155) is well rounded while GC Lisp ($465) supports syntax closer to "Common LISP." Or Prolog-86 ($125).

PROFILER-86 - find where any program spends most of its time quickly, easily. DOC nicely discovers theory, key issues. MSDOS. $125

For a catalog, comparisons, prices, or for an info packet on AI, or Editors, "C," BASIC, PASCAL, FORTRAN, or COBOL—or just for straight answers—

CALL TOLL FREE 800-421-8006

THE PROGRAMMER'S SHOP™

The programmer's complete source for software, services and answers

128-L Rockland Street, Hanover, MA 02339 In Mass.: **800-442-8070** or 617-826-7531

CIRCLE 52 ON READER SERVICE CARD

THE FORTH SOURCE™

MVP-FORTH

Stable - Transportable - Public Domain - Tools

You need two primary features in a software development package . . . a stable operating system and the ability to move programs easily and quickly to a variety of computers. MVP-FORTH gives you both these features and many extras. This public domain product includes an editor, FORTH assembler, tools, utilities and the vocabulary for the best selling book "Starting FORTH". The Programmer's Kit provides a complete FORTH for a number of computers. Other MVP-FORTH products will simplify the development of your applications.

MVP Books – A Series

- ☐ **Volume 1,** *All about FORTH* by Haydon. MVP-FORTH glossary with cross references to fig-FORTH, *Starting FORTH* and FORTH-79 Standard. 2nd Ed. $25
- ☐ **Volume 2,** *MVP-FORTH Assembly Source Code.* Includes CP/M®, IBM-PC®, and APPLE® listing for kernel $20
- ☐ **Volume 3,** *Floating Point Glossary* by Springer $10
- ☐ **Volume 4,** *Expert System* with source code by Park $25
- ☐ **Volume 5,** *File Management System* with interrupt security by Moreton $25

MVP-FORTH Software – A Transportable FORTH

- ☐ **MVP-FORTH Programmer's Kit** including disk, documentation Volumes 1 & 2 of MVP-FORTH Series (*All About FORTH*, 2nd Ed. & Assembly Source Code), and *Starting FORTH*. Specify ☐ CP/M, ☐ CP/M 86, ☐ CP/M+, ☐ APPLE, NEW ☐ IBM PC/XT/AT, ☐ MS-DOS, ☐ Osborne, ☐ Kaypro, ☐ H89/Z89, ☐ Z100, ☐ TI-PC, ☐ MicroDecisions, ☐ Northstar, ☐ Compupro, ☐ Cromenco, ☐ DEC Rainbow, NEW ☐ NEC 8201, ☐ TRS-80/100, ☐ HP 110, ☐ HP 150, ☐ STM PC $150
- ☐ **MVP-FORTH Enhancement Package** for IBM-PC/XT Programmer's Kit. Includes full screen editor, MS-DOS NEW file interface, disk, display and assembler operators. $110
- ☐ **MVP-FORTH Cross Compiler** for CP/M Programmer's Kit. Generates headerless code for ROM or target CPU $300
- ☐ **MVP-FORTH Meta Compiler** for CP/M Programmer's kit. Use for applicatons on CP/M based computer. Includes public domain source $150
- ☐ **MVP-FORTH Fast Floating Point** Includes 9511 math chip on board with disks, documentation and enhanced virtual MVP- FORTH for Apple II, II+, and IIe. $450
- ☐ **MVP-FORTH Programming Aids** for CP/M, IBM or APPLE Programmer's Kit. Extremely useful tool for decompiling, callfinding, and translating. $200
- ☐ **MVP-FORTH PADS (Professional Application Development System)** for IBM PC, XT or PCjr or Apple II, II+ or IIe. An integrated system for customizing your FORTH programs and applications. The editor includes a bi-directional string search and is a word processor specially designed for fast development. PADS has almost triple the compile speed of most FORTH's and provides fast debugging techniques. Minimum size target systems are easy with or without heads. Virtual overlays can be compiled in object code. PADS is a true professional development system. Specify Computer. $500
- ☐ **MVP-FORTH Floating Point & Matrix Math** for IBM with 8087 or Apple with Applesoft on Programmer's Kit or PADS. $85
- ☐ **MVP-FORTH Graphics Extension** for IBM or Apple on Programmer's Kit or PADS. $65
- ☐ **MVP-FORTH MS-DOS** file interface for IBM PC PADS $80
- ☐ **MVP-FORTH Expert System** for development of knowledge-based programs for Apple, IBM, or CP/M. $100

FORTH CROSS COMPILERS Allow extending, modifying and compiling for speed and memory savings, can also produce ROMable code. Specify CP/M, 8086, 68000, IBM, Z80, or Apple II, II+ $300

Ordering Information: Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard, American Express. COD's $5 extra. Minimum order $15. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping by Air: $5 for each item under $25, $10 for each item between $25 and $99 and $20 for each item over $100. All prices and products subject to change or withdrawal without notice. Single system and/or single user license agreement required on some products.

FORTH DISKS

FORTH with editor, assembler, and manual.

- ☐ **APPLE** by MM, 83 $100
- ☐ **ATARI®** valFORTH $60
- ☐ **CP/M** by MM, 83 $100
- ☐ **HP-85** by Lange $90
- ☐ **HP-75** by Cassady $150
- ☐ **IBM-PC** by LM, 83 $100
- ☐ **NOVA** by CCI 8" $175
- ☐ **Z80** by LM, 83 $100
- ☐ **8086/88** by LM, 83 $100
- ☐ **68000** by LM, 83 $250
- ☐ **VIC FORTH** by HES, VIC20 cartridge $50
- ☐ **C64** by HES Commodore 64 cartridge $40
- ☐ **Timex** by HW $25

Enhanced FORTH with: F-Floating Point, G-Graphics, T-Tutorial, S-Stand Alone, M-Math Chip Support, MT-Multi-Tasking, X-Other Extras, 79-FORTH-79, 83-FORTH-83.

- ☐ **APPLE** by MM, F, G, & 83 $180
- ☐ **ATARI** by PNS, F,G, & X. $90
- ☐ **CP/M** by MM, F & 83 $140
- ☐ **Multi-Tasking FORTH** by SL, CP/M, X & 79 $395
- ☐ **TRS-80/I or III** by MMS F, X, & 79 $130
- ☐ **Timex** by FD, tape G,X, & 79 $45
- ☐ **C64** by ParSec. MVP, F, 79, G & X $96
- ☐ **Victor 9000** by DE,G,X $150
- ☐ **Extensions** for LM Specify IBM, Z80, or 8086
  - ☐ Software Floating Point $100
  - ☐ 8087 Support (IBM-PC or 8086) $100
  - ☐ 9511 Support (Z80 or 8086) $100
  - ☐ Color Graphics (IBM-PC) $100
  - ☐ Data Base Management $200
- ☐ **fig-FORTH Programming Aids** for decompiling, callfinding, debugging and translating. CP/M, IBM-PC, Z80 or Apple. $200

FORTH MANUALS, GUIDES & DOCUMENTS

- NEW ☐ **Thinking FORTH** by Leo Brodie, author of best selling "Starting FORTH" $16
- ☐ **ALL ABOUT FORTH** by Haydon. See above. $25
- ☐ **FORTH Encyclopedia** by Derick & Baker $25
- ☐ **The Complete FORTH** by Winfield $16
- ☐ **Understanding FORTH** by Reymann $3
- ☐ **FORTH Fundamentals,** Vol. I by McCabe $16
- ☐ **FORTH Fundamentals,** Vol. II by McCabe $13
- ☐ **FORTH Tools, Vol.1** by Anderson & Tracy $20
- ☐ **Beginning FORTH** by Chirlian $17
- ☐ **FORTH Encyclopedia Pocket Guide** $7
- ☐ **And So FORTH** by Huang. A college level text. $25
- ☐ **FORTH Programming** by Scanlon $17
- ☐ **FORTH on the ATARI** by E. Floegel $8
- ☐ **Starting FORTH** by Brodie. Best instructional manual available. (soft cover) $19
- ☐ **Starting FORTH** (hard cover) $23
- ☐ **68000** fig-Forth with assembler $25
- ☐ **1980 FORML Proc.** $25
- ☐ **1981 FORML Proc** 2 Vol $40
- ☐ **1982 FORML Proc.** $25
- ☐ **1981 Rochester FORTH Proc.** $25
- ☐ **1982 Rochester FORTH Proc.** $25
- ☐ **1983 Rochester FORTH Proc.** $25
- ☐ **A Bibliography of FORTH References, 1st Ed.** $15
- **The Journal of FORTH Application & Research**
  - ☐ **Vol. 1, No. 1** $15
  - ☐ **Vol. 1, No. 2** $15
- ☐ **METAFORTH** by Cassady $30
- ☐ **Threaded Interpretive Languages** $23
- ☐ **Systems Guide to fig-FORTH** by Ting $25
- ☐ **FORTH Notebook** by Ting $25
- NEW ☐ **Invitation to FORTH** $20
- ☐ **PDP-11 User Man.** $20
- ☐ **FORTH-83 Standard** $15
- ☐ **FORTH-79 Standard** $15
- ☐ **FORTH-79 Standard Conversion** $10
- ☐ **Tiny Pascal fig-FORTH** $10
- ☐ **NOVA fig-FORTH** by CCI Source Listing $25
- ☐ **NOVA** by CCI User's Manual $25

☐ **Installation Manual for fig-FORTH,** $15

Source Listings of fig-FORTH, for specific CPU's and computers. The Installation Manual is required for implementation. Each $15

☐ 1802 ☐ 6502 ☐ 6800 ☐ AlphaMicro ☐ IBM
☐ 8080 ☐ 8086/88 ☐ 9900 ☐ APPLE II
☐ PACE ☐ 6809 ☐ NOVA ☐ PDP-11/LSI-11
☐ 68000 ☐ Eclipse ☐ VAX ☐ Z80

MOUNTAIN VIEW PRESS, INC.

PO BOX 4656 MOUNTAIN VIEW, CA 94040 (415) 961-4103

The complete ZCPR3 documentation is to be released in the near future.

The installation of ZCPR3 requires a good working knowledge of CP/M and your system's BIOS. It also requires that the installer be very familiar with assembly language programming. The installation of this package is not a job for the novice.

To install ZCPR3 you will need the Digital Research Macro Assembler MAC, an editor, and the Digital Research debugger ZSID. Robert Van Valzah's public domain RELS.UTL is needed to install the command file processor ZEX.COM. The latter utility was included on a update disk sent later. Another public domain utility that is used in the installation examples but not provided is Ron Fowler's MLOAD.COM. While this utility is not mandatory its inclusion would have been nice.

As with any project, the first step is planning and organization. You must first decide which of the many features available you will want to implement. The sampler manual was some help in my selection of features. Having the full ZCPR3 documentation should make this much simpler.

It may be necessary to do a few experimental assemblies as the selection of features will effect the size of the various modules. Remember that the more features implemented in memory the smaller the TPA. My goal was to enable as many features as possible and still retain at least 48K of TPA. My reason for this was that most commercial CP/M software is written to also run under MP/M which will only support up to 48K per user bank.

After some experimentation I drew up the above memory map to use as a guide for setting up the system base library module (Z3BASE.LIB). Once this file is edited and the features of the other library files selected, assembly of the individual modules may begin.

The CCP replacement (ZCPR3) and the modified BIOS hex files need to be placed into the CP/M system image file created by *SYSGEN* or *MOVCPM*.

After the external modules are assembled, the resultant hex files must be converted to object modules. The *MLOAD* utility can be used for this. If you don't have this utility, Digital Research's DDT.COM can be used. If *DDT* is used, you will need to compute the offset required to move the hex file to location 100H, making it a saveable file.

One of the modules created from the preceding process is the system environment module. Using this module and the ZCPR3 installation program Z3INS.COM, all of the ZCPR3 utilities can now be installed.

You now need only create a new system disk using *SYSGEN* and copy the needed modules and utilities to it. The resultant disk should now be a bootable system disk. Booting this disk will not, however, cause all of the needed modules to be loaded. You need to create a startup program with the ALIAS.COM utility to evoke the module loader program LDR.COM to load the various modules. As can be seen, this is not a project for the novice.

But . . . take heart novice, you have been saved! Late into this review I received yet another disk from Echelon. This disk contains a completely automatic installation program called Z3—DOT—COM. The disk includes most but not all of the ZCPR3 utilities and several relocatable installation modules. All that is required to install ZCPR3 with this disk is to first format and sysgen a new CP/M disk, transfer all the files from the Z3—DOT—COM disk and run the supplied submit file with SUBMIT.COM.

I set up a disk for my second test system, a TRS-80 Model II with Pickels & Trout CP/M, and ran the submit file. The installation took less than 4 min from the time the submit file was started. At the end of the installation the ZCPR3 utility *TCSELECT* is loaded and a menu of terminals is presented for you to select from. If your terminal isn't included in this menu you can exit the program, proceed with the installation, and at the conclusion use the *TCMAKE* utility to configure your own terminal descriptors.

If you copy only the contents of the Z3—DOT—COM disk to your new disk you will find that the install program will report missing files during the installation. If you wish to install these files you will need to copy them from the original ZCPR3 source disks and install them with Z3INS.COM, using the supplied ZCPR3.INS and newly created Z3.ENV file.

Once the installation is complete, two new files will be present on your disk—Z3.COM and Z3X.COM. The Z3.COM program is the ZCPR3 loader that brings up ZCPR. The Z3X.COM program is used to exit ZCPR and return to standard CP/M.

The Z3—DOT—COM version of ZCPR3 is pre-configured to take advantage of all but the redirectable I/O features of ZCPR3. The configuration can be changed by patching the Z3.COM file, but this again takes us out of the novice area of expertise.

The only real drawback I found with the Z3—DOT—COM installed version is that many of the Pickels & Trout system utilities wouldn't run correctly while ZCPR3 is active. The reason for this is that the BIOS is not truly located where the utilities think it is.

The BIOS is the one portion of the operating system that is not relocated. Instead, a copy of the BIOS jump table is placed at the end of the relocated BDOS. This jump table directs BIOS calls to the real BIOS. Some systems, such as Pickels & Trout CP/M, keep system configuration information in the BIOS module. Utilities designed for this system may expect to find this data at some displacement from the base of BIOS. This is not as big a problem as it may first seem to be since real CP/M can be re-entered at any time by running the Z3X program.

It is also interesting to note that when the installation program makes a copy of the BIOS jump table, it copies the entire jump table and not just the normal 17 jumps required by standard CP/M. This is important because some system BIOS modules have extended capabilities built-in.

Besides the 10 ZCPR3 disks supplied, four more disks with systems library source files were also supplied. These are the modules that make up the SYSLIB.REL file used by most of the ZCPR3 utilities. While no hard documentation was provided, there are 21 help files on one of the four disks.

However, the source code was not provided for Z3LIB.REL (which contains ZCPR3 specific routines), and VLIB.REL (which contains routines to take advantage of Z3TCAP environment). The help file Z3TCAP.HLP gives an overview of VLIB.REL usage but refers you to the VLIB help file for specific information. Unfortunately, this file was not to be found on any of the distribution diskettes. An addendum file is supplied for those who already have documentation for ZCPR2 SYSLIB, and a READ.ME file refers you to other ZCPR2 documentation for the ZCPR specific routines in the Z3LIB.REL file. Since Echelon is looking forward to programmers writing ZCPR3 programs, I would hope the company intends to fill the documentation void.

The method of moving around under ZCPR3 is a vast improvement to the conventional CP/M method. Under CP/M, to log into a Drive/User area you first have to log onto the desired drive and then, using the *USER* command, log into the desired user area. Under ZCPR3, depending on the configuration options you selected, you can log into a Drive/User area by simply typing the drive letter followed by the user number and a colon. The following two examples illustrate the difference between ZCPR3 and CP/M in logging into drive B: user area 12, the area in which we have, for the sake of demonstration, our Pascal compiler.

First the entries required by CP/M:

```
A>B:
B>USER 12
B>
```

Now the ZCPR3 entry:

```
A0:BASE>B12:
B12:PASCAL>
```

Notice the difference in the prompts displayed. While CP/M only displays the letter of the logged in drive, ZCPR3 can display the drive and user identity as well as directory name we have chosen for this area.

Under ZCPR3 there are two more ways we could have logged into this area. Both of these methods use the named directory entry for the Drive/User area. The first method is demonstrated as follows:

```
A0:BASE>PASCAL:
B12:PASCAL>
```

The other method is to use the change directory utility CD.COM as follows:

```
A0:BASE>CD PASCAL:
     Logging Into B12:PASCAL:
B12:PASCAL>
```

The advantage of using the *CD* utility is that when CD.COM logs into a user area it looks for a program called ST.COM. If found, ST.COM is executed, if not, *CD* just logs you into the specified user area.

The program ST.COM is a startup program that the user can create with the *ALIAS* utility. The startup program can be used for a number of purposes. For example, a new named directory could be loaded and new paths invoked or one of the ZCPR3 menu systems could be invoked. As can be seen, this capability lends itself to the construction of a very powerful turnkey system.

Another advantage of using directory names as the method of access is that each named directory entry can have an associated password assigned to it. When ZCPR3 detects a directory entry in the command line, it checks to see if a password has been assigned. If one has, ZCPR3 will prompt the user for the password. If the user's response matches, the operation proceeds, if not, ZCPR3 will not allow access to that area.

Named directories are created with the MKDIR.COM utility and loaded into memory with the LDR.COM utility. All of the ZCPR3 utilities support the use of directory names as Drive/User identifiers.

Besides passwords there is another form of system protection under ZCPR3—the wheel facility. The wheel facility is the systems operator's safeguard against unauthorized access of privileged user areas and system utilities. A flag called the "wheel byte" resides somewhere in memory; its location is determined at installation time. When off, this flag will prevent the execution of several of the ZCPR3 utilities.

System resident commands are those commands that are located in the ZCPR3 command processor or in the resident command package. For a list of commands that can be made resident, call the *COMPUTER LANGUAGE* Bulletin Board Service or dial into CompuServe and type "Go CLM." A list of commands that reside in the resident flow control package are printed in this review (Table 1).

The use of the supplied ZCPR3 utilities is fairly well covered in the supplied help files. In addition, most of the utilities will print their own help information if they see a double / on the command line. A brief description of most of the ZCPR3 utilities is on the BBS or CompuServe.

After many hours of trying out the different commands and utilities of ZCPR3, I found no obvious bugs or glitches in the system. But just running systems utilities isn't the true test of a system. So I spent quite a few more hours testing the system with off-the-shelf CP/M software and achieved the same results. Because there are so many utilities and such a large collection of help files, I highly recommend the use of a hard disk system. I feel that ZCPR3 goes a long way in improving the CP/M work environment. My only reservation is the current state of the available documentation.

By Dennis L. Wright

IF T
Gives a true condition if the flow state is set true

IF F
Gives a true condition if the flow state is set false

IF EMPTY dir:filename.typ
Sets the flow state to true if the specified file is empty or does not exist

IF ERROR
Sets the flow state to true if the ZCPR3 system error flag is SET

IF EXIST dir:filename.typ
Sets the flow state to true if the specified file exists

IF INPUT
The user is prompted for input and if he responds with T, Y, <CR>, or <SP> the flow state is set to true

IF NULL filename
If the second filename in a command is blank the flow state is set true

IF n (register value) val (value to compare to)
If the ZCPR3 register specified by (n) equals the value specified by (val) the flow state is set to true

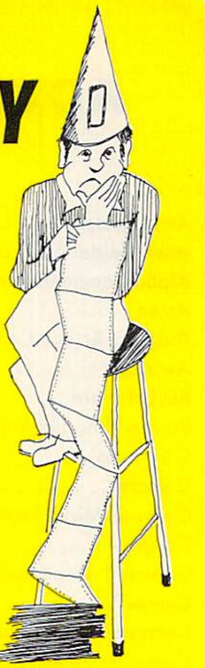
IF TCAP
Sets the flow state to true if the ZCPR3 TCAP contains a terminal definition

IF WHEEL
Sets the flow state to true if the wheel byte is set

IF filename1 = filename2
Sets the flow state to true if the two specified file names are the same

These conditions can be negated by preceding the condition with a tilda (~). If the condition is FALSE, the flow state is set to TRUE, and vice-versa.

Table 1.



WHY JOHNNY CAN'T READ HIS OWN CODE

Johnny's A Good Programmer, Even Brilliant, But– Johnny works in 8080/Z80 assembly language, with a conventional assembler. That can make yesterday's brilliance today's garble, a maze of mnemonics and a jumble of meaningless labels. Johnny's program is less than self-explanatory—even for Johnny.

Johnny *could* read his own code if he used SMAL/80—the *superassembler*—and so can you. SMAL/80 boosts your program's clarity and your productivity by giving you:

■ Familiar algebraic notation in place of cryptic mnemonics—"A = A–3" for example, instead of "SUI 3" (if you know BASIC or Pascal, you *already* know SMAL/80)

■ Control structures like BEGIN...END, LOOP...REPEAT WHILE, and IF...THEN...ELSE... to replace tangled branches and arbitrary label names (eliminating up to 90% of labels with *no* overhead imposed)

■ Complete control over your processor—because SMAL/80 is a true assembler, it doesn't reduce execution speed or burden your program with its own runtime routines.

SMAL/80, the assembler that handles like a high-level language, lets you do it right the first time, and lets you read and understand your work afterward—the next day or a year later. Users say SMAL/80 has doubled and even tripled their output of quality code. But don't take our word for it—***TRY IT!***

Use SMAL/80 for 30 days. If you're not *completely* satisfied with it—for *any* reason—return the package for a full refund.

SPECIAL BONUS: Order before Dec. 31, 1984, and get *Structured Microprocessor Programming*—a $25 book ***FREE!***

SMAL/80 for CP/M-80 systems (*all* CP/M disk formats available—please specify); produces 8080/8085 and Z80 code. Now supports Microsoft .REL. **ONLY $149.95**

SMAL/80 for CP/M-80 systems, 8080/8085 output only. **SAVE $20: $129.95**

NEW! SMAL/80X65—for Apple II and IIe (requires Z80 card and CP/M); produces Z80 *and* 6502 object code. **$169.95**

Mastercard
Visa
C.O.D.'s

SMAL/80
CHROMOD ASSOCIATES
(201) 653-7615
1030 Park Ave. Hoboken, N.J. 07030

We pay shipping on prepaid orders

CIRCLE 10 ON READER SERVICE CARD

ADVERTISER INDEX

	PAGE NO.	CIRCLE NO.
Alcor Systems	20	1
Allen Gelder Software	64	28
Alpha Computer Service	50	2
Atron	2	4
Austin E. Bryant Consulting	50	7
Awareco	32	3
BD Software	22	5
Borland International	6&7	6
C Systems	38	16
C Ware	39	18
Carousel Micro Tools	24	8
Catspaw	64	9
Chromod Associates	79	10
CompuPro	Cover IV	12
Computer Innovations	45	11
Computer Resources of Waimea	19	14
DWB Associates	64	20
Datalight	15	19
Ecosoft	59	22
Greenleaf Software	52	29
HSC, Inc.	10	31
Introl Corp.	11	32
Korsmeyer Electronics Design Inc.	4	34
Laboratory Microsystems, Inc.	34	35
Lattice Inc.	40	36
mbp Software & Systems Technology	23	39
Megamax, Inc.	14	13
MicroMotion	20	40
Microcompatibles	24	15
Mountain View Press	77	43
Next Generation Systems	62	45
Northwest Computer Algorythms	52	46
Parsec Research, Inc.	15	49
Plum Hall	49	50
Poor Person Software	53	51
ProCode	34	53
Programmer's Connection	75	54
Programmer's Shop	76	52
Programmer's Shop	60	17
QCAD	14	23
Quest Research	Cover III	55
RR Software	Cover II	58
Raima Corporation	40	24
Rational Systems	16	56
SLR Systems	4	59
Software Horizons	33	25
Software Toolworks	50	26
Solution Systems	42	60
Solution Systems	42	61
Solution Systems	54	27
Spruce Technology	63	33
Steve Rank Inc.	75	37
Summit Software	46	62
System Engineering Tools	53	64
The Code Works	19	42
Thunder Software	64	65
UniPress	46	38
Western Ware	64	67
Wordtech Systems	1	41
Workman & Associates	75	68

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

ORDER THE PREMIER ISSUE OF COMPUTER LANGUAGE

The first issue of *COMPUTER LANGUAGE* was a great success and nearly sold out in just one month. We still have a few copies of this collectors edition available now. Just fill out this coupon and mail it back with $4.00 per issue.

Please send me _____ copies of COMPUTER LANGUAGE's premier issue at $4.00 per issue, $_____ Total

NAME ____________________

COMPANY ____________________

ADDRESS ____________________

CITY, STATE, ZIP ____________________

Send payment and coupon to: COMPUTER LANGUAGE
Premier Issue
131 Townsend St.
San Francisco, CA 94107

ADVERTISE in the February issue of COMPUTER LANGUAGE

Reservation Deadline:
December 3rd

Contact:
Carl Landau or Jan Dente
Computer Language
131 Townsend Street
San Francisco, CA 94107
(415) 957-9353

SUBSCRIBE TODAY!

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** today for only $24.00 — over 33% savings off the single copy price.

☐ Yes, start my Subscription to COMPUTER LANGUAGE today. The cost is only$24.00 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $39.00.

☐ Payment enclosed ☐ Bill me

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Canada orders $30.00 per year. Outside the U.S., $36.00/year for surface mail or $54.00/year for airmail.

B1N4

SUBSCRIBE TODAY!

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** today for only $24.00 — over 33% savings off the single copy price.

☐ Yes, start my Subscription to COMPUTER LANGUAGE today. The cost is only$24.00 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $39.00.

☐ Payment enclosed ☐ Bill me

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Canada orders $30.00 per year. Outside the U.S., $36.00/year for surface mail or $54.00/year for airmail.

B1N4

SUBSCRIBE TODAY!

COMPUTER LANGUAGE

Subscribe to **COMPUTER LANGUAGE** today for only $24.00 — over 33% savings off the single copy price.

☐ Yes, start my Subscription to COMPUTER LANGUAGE today. The cost is only$24.00 for 1 year (12 issues).

☐ I want to increase my savings even more — send me 2 years (24 issues) of COMPUTER LANGUAGE for only $39.00.

☐ Payment enclosed ☐ Bill me

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Please allow 6-8 weeks for delivery of first issue. Foreign orders must be prepaid in U.S. funds. Canada orders $30.00 per year. Outside the U.S., $36.00/year for surface mail or $54.00/year for airmail.

B1N4

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

COMPUTER
LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

READER SERVICE CARD

FREE!

ree information from the advertisers of
OMPUTER LANGUAGE.

. Please fill in your name and address on the card (one person to a card).
. Answer questions 1-3.
. Circle the numbers that correspond to the advertisements you are interested in.

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Country ________ Telephone number ____________________

November issue. Not good if mailed after March 31, 1985.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Please complete these short questions:

1. I obtained this issue through:
   - ☐ Subscription
   - ☐ Computer Store
   - ☐ Retail outlet
   - ☐ Passed on by associate
   - ☐ Other ____________
2. Job Title ____________________
3. The 5 languages that I am most interested in reading about (list in order of importance).

____

____

Comments ____________________

____

____

Attn: Reader Service Dept.

1/3

READER SERVICE CARD

FREE!

ree information from the advertisers of
OMPUTER LANGUAGE.

. Please fill in your name and address on the card (one person to a card).
. Answer questions 1-3.
. Circle the numbers that correspond to the advertisements you are interested in.

Name ____________________

Company ____________________

Address ____________________

City, State, Zip ____________________

Country ________ Telephone number ____________________

November issue. Not good if mailed after March 31, 1985.

Circle numbers for which you desire information.

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98
9	19	29	39	49	59	69	79	89	99
10	20	30	40	50	60	70	80	90	100

Please complete these short questions:

1. I obtained this issue through:
   - ☐ Subscription
   - ☐ Computer Store
   - ☐ Retail outlet
   - ☐ Passed on by associate
   - ☐ Other ____________
2. Job Title ____________________
3. The 5 languages that I am most interested in reading about (list in order of importance).

____

____

Comments ____________________

____

____

Attn: Reader Service Dept.

1/3

Editorial Response Card

eader suggestions

We want to hear your comments and suggestions about this issue of **COMPUTER LANGUAGE.** Your reader feedback will enable us to provide you with the information you want. Thank you for your help!

Comments: ____________________

____

____

☐ Yes, I have an idea for a manuscript: ____________________

____

☐ Yes, I'm interested in reviewing technical manuscripts.

☐ Yes, I'm interested in reviewing software.

Name: ____________________

Company: ____________________

Address: ____________________

City, State, Zip: ____________________

Phone Number: ____________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 27346 PHILADELPHIA, PA USA

POSTAGE WILL BE PAID BY

COMPUTER LANGUAGE

P.O. BOX 11747
PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 27346 PHILADELPHIA, PA USA

POSTAGE WILL BE PAID BY

COMPUTER LANGUAGE

P.O. BOX 11747
PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 22481 SAN FRANCISCO, CA USA

POSTAGE WILL BE PAID BY

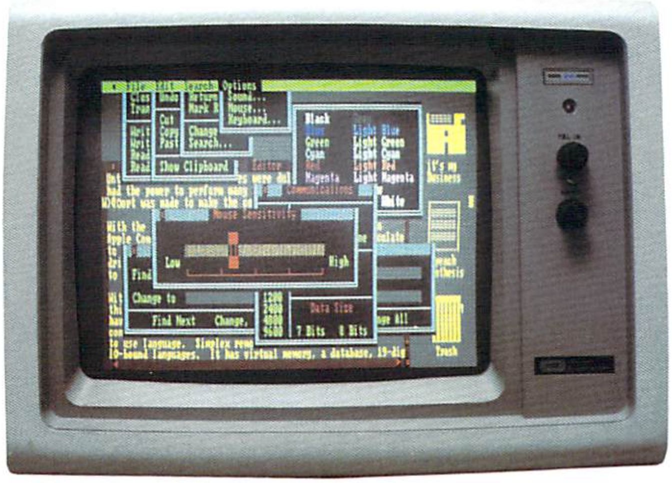
COMPUTER LANGUAGE

2443 FILLMORE STREET • SUITE 346
SAN FRANCISCO, CA 94115

We thought about calling it MacSimplex . . . after all it makes your IBM®PC behave like a Macintosh™ and much more . . .

and with over two years in the making, the Simplex Database Management System has features like 32-megabyte virtual memory and the most powerful networked/relational database in the microcomputer industry. Simplex was designed around how you think and the Macintosh way, so that you can use your favorite mouse to handle those mundane tasks like menu selection and data manipulation. And, if you don't have a mouse, you can use our keyboard mouse simulator, MouSim™.

Pop-up and pull-down menus, dialog and alert boxes are not just added features, they are the heart of the Simplex way. In addition, Simplex gives you both a software and a hardware floating point capability, each with 19-digit accuracy. It permits login, password, privilege, and can be used on a local area network. Simplex has full communications and a remote or local printer spooler. Above all, Simplex is modular and grows with you! Simplex also has a full-featured, English-like language which is simple to use.



You can't buy Simplex™, but it is now available as an integral part of *it's my* ***Business***™ and will be used by *it's my* ***Word***™, *it's my* ***Graphics***™, . . .

Businessmen! *it's my* ***Business*** will revolutionize the way that you handle your business. It saves time, money, and standardizes your system for all who use it. *it's my* ***Business*** comes with applications like accounting, interoffice or intraoffice mail, editing, invoicing, inventory managment, mail list, calendar, scheduler, forms and more. You can modify each of these to create applications specifically designed for you... maybe we should have called it "it's your Business".

Professionals! *it's my* ***Business*** has over 200 pages of examples and demonstrations to show you how to solve your everyday professional problems. And if these examples aren't enough, we give you a complimentary one-year subscription to Questalk™, our hands-on Simplex applications magazine.

System integrators and consultants, beware! If you are not using *it's my* ***Business*** with Simplex to solve your problems, don't be surprised when more novice programmers solve that complex math, industrial engineering, or business problem faster. We think that you can cut your concept-to-development time by an order of magnitude!

it's my ***Business*** (includes *it's my* ***Editor***) - $695.00
it's my ***Business*** Demo Disk - $20.00
it's my ***Editor*** - $100.00.

Quest Research software is available through your local computer store or through mail order from Quest Software Corporation at (205) 539-8086. 303 Williams Avenue, Huntsville, AL 35801.

Value added resellers and dealers please contact Quest Research, Incorporated at (800) 558-8088. 303 Williams Avenue, Huntsville, AL, 35801.

Quest™

Quest Research Inc.

IBM is a registered trademark of International Business Machines. Macintosh is a trademark of Apple Corporation. *it's my* ***Business***, *it's my* ***Word***, *it's my* ***Graphics***, *it's my* ***Editor***, *it's my* ***Home***, *it's my* ***Voice***, *it's my* ***Ear***, *it's my* ***Statistics***, Simplex, MouSim, Questalk, and the Quest logo are trademarks of Quest Research, Incorporated.

CIRCLE 55 ON READER SERVICE CARD

HERE TODAY HERE TOMORROW

When buying a computer, you can't limit yourself to just satisfying today's needs. **The best value in a system comes from its productivity . . . both for today and tomorrow.** CompuPro's System 816™ computer has that value. With all the power and capacity to handle your needs now and down the road.

System 816's longevity stems from top quality components . . . high storage capacity . . . the flexibility to handle a large variety of applications . . . and the speed to get the job done fast. Upgrading is easy, and when it's time to expand from single to multi-user operation, it's as simple as plugging in boards and adding terminals. Your system grows as you grow.

CompuPro also provides a library of the most popular software programs with your system and because it's CP/M® based, you have more than 3,000 other programs to choose from.

Even our warranty is for today and tomorrow. It spans 365 days — and includes the additional security of Xerox Americare™ on-site service nationwide for designated systems.*

What's more, CompuPro is one company you can count on to be around tomorrow. For more than ten years we've been setting industry standards, increasing productivity and solving problems.

For a free copy of our business computer buyer's primer, and the location of the Full Service CompuPro System Center nearest you, call **(415) 786-0909 ext. 206.**

CompuPro's System 816. The computer that's just as essential tomorrow as it is today.

CompuPro®
A *GODBOUT* COMPANY
3506 Breakwater Court, Hayward, CA 94545

*Available from Full Service CompuPro System Centers and participating retailers only.
System 816 and The Essential Computer are trademarks of CompuPro.
CP/M is a registered trademark of Digital Research Inc. Americare is a trademark of Xerox Corporation.
System 816 front panel design shown is available from Full Service CompuPro System Centers only. ©1984 **CompuPro**

The Essential Computer™

CIRCLE 12 ON READER SERVICE CARD